READY-PLAYER-ONE

Rapport du groupe Harpon

Martin Lehoux, Pierre Biret
Sacha Seksik, Loïc Audoin

December 11, 2018

ABSTRACT

Le projet "Ready player one", constitué de deux équipes (Harpon et Eponge) a pour but l'apprentissage par un réseau de neurones d'une stratégie gagnante au jeu "Pong" sur Atari.

L'interêt du projet étant la compréhension des mécanismes des réseaux de neurones par l'ensemble des étudiants, ce projet sera donc constitué de plusieurs parties qui mèneront à terme à la réalisation du projet.

1 RÉALISATION DU PERCEPTRON

La première étape de l'apprentissage de notre groupe est le fonctionnement d'un neurone. Une feuille de calcul a donc été mise en place pour calculer à la main une itération de l'apprentissage d'un réseau de neurones de la fonction XOR à deux entrées avec rétropropagation, après visualisation par tous les étudiants du groupe d'une vidéo très bien réalisée sur le fonctionnement des réseaux neuronaux.

La mise en place de cette feuille de calcul a mis en évidence la complexité des calculs effectués par les réseaux de neurones, ainsi que l'impossibilité d'effectuer tous les calculs à la main. Toutefois ce premier exemple de réseaux a permis à l'ensemble de l'équipe de comprendre comment se comportait un réseau de neurones à plusieurs couches, et leur a permis de se lancer dans le vif du sujet.

L'étape suivante fut donc la programmation en Python d'un réseau de neurones. Les modules matplotlib et numpy ont été utilisés pour les calculs matriciels, améliorant la vitesse de calcul. Chaque équipe a tenté lors de cette étape de coder sa version d'un réseau d'apprentissage du XOR, et malgré la simplicité apparente de cette tâche, les résultats ont divergé.

Le premier constat fut le suivant : L'assurance de la convergence de l'erreur vers 0 ne se fait que lorsque la couche intermédiaire du réseau a au moins 4 neurones. (Figures : réseau de neurones 2-2-1 , divergence, réseau de neurones 2-4-1, convergence) Afin de s'assurer que les calculs et les résultats des deux équipes étaient bien les mêmes, la tâche leur fut confiée de reproduire à l'identique deux réseaux, de les implémenter dans leurs versions respectives codées sous Python, et de comparer les résultats, en prenant en compte l'intervalle de confiance de la valeur de l'erreur.

La suite était alors la réalisation d'un réseau de neurones pouvant travailler sur la base de de données MNIST des chiffres manuscrits, et sachant à terme reconnaître les chiffres. Le premier souci en terme de travail d'équipe a été rencontré à ce moment-là : deux membres du groupe avaient chacun écrit une version de l'algorithme d'apprentissage, et après discussion tendues, les deux membres ont travaillé ensemble à la réalisation du perceptron.

Les premiers résultats différaient énormément des résultats théoriques, surtout en terme de vitesse de calcul et en taux d'apprentissage : alors qu'il devrait être de l'ordre de 10^{-4} pour obtenir une convergence précise et suffisante, nous pouvions monter jusqu'à 10^{-1} voire 1, et toujours converger assez lentement. Ce problème restera dans notre code sans en trouver de réelle raison pendant environ 1 mois.

Quant à la vitesse de calcul, elle était grandement affectée par le premier choix que nous avions fait de conserver les formules itératives des calculs de valeurs de neurones. L'équipe Eponge, en parallèle, avait un temps de calcul de l'ordre de la seconde, tandis que nous avions un temps de calcul de l'ordre de l'heure. Le passage de la forme itérative à la forme matricielle, pour le calcul des valeurs des réseaux de neurones optimisa grandement le temps de calcul, et est une preuve de l'incroyable efficacité des modules de calcul matriciel de Python.

Toutefois les valeurs des erreurs finales étaient des valeurs cohérentes, et nous avions donc décidé de travailler sur d'autres facteurs, tels que la taille des batch (qui n'étaient alors que de 1), et l'initialisation du réseau.

Les travaux de LeCun affirmaient que l'initialisation des valeurs des poids et des biais des neurones étaient à rendre aléatoires, et cela fut rapidement confirmé en voyant les courbes d'apprentissage des réseaux neuronaux avec des poids tous initialisés à 0.

La taille du batch, toutefois, était une question différente : la conclusion qui nous est venue est la suivante : le meilleur compromis entre stabilité et précision de l'erreur finale est obtenu pour des batch de taille entre 16 et 32. Ces résultats sont appuyés par LeCun.

Tous nos calculs ont alors gagné en temps grâce à la machine virtuelle fournie par nos professeurs encadrants avec l'aide de l'école. L'un des membres de l'équipe disposait également d'une carte graphique NVIDIA 960M, mais au vu de la com-

plexité d'utilisation du CUDA NVIDIA servant aux calculs mathématiques sur carte graphique, la décision a été prise de continuer sur la machine virtuelle.

Le dernier point concernant les perceptrons est le point de la vitesse de convergence : nous avions, depuis le début, utilisé la fonction d'activation sigmoïde basique Le changement de la fonction d'activation à un tanh pondéré nous semblait au premier abord inefficace, voire semblait empirer les résultats. En effet, l'erreur de la reconnaissance des chiffres de la base MNIST ne convergeait même pas lors de nos tests avec la nouvelle fonction d'activation.

En réalité, l'extrême efficacité de cette fonction, suggérée encore une fois par LeCun, associée à nos taux d'apprentissage bien trop élevés, empêchaient la convergence du réseau de neurones. Quelques jours plus tard, le groupe se rendit compte que la même fonction d'activation, avec des taux extrêmement bas, convergeaient à une vitesse fulgurante, voire trop rapidement. Nous sommes donc passés d'un problème de convergence trop lente à une convergence trop rapide.

Cette nouvelle efficacité nous a permis de déterminer une toute dernière caractéristique de la base MNIST : sa taille et sa diversité sont si grandes qu'il nous a été impossible d'atteindre l'état de surapprentissage.

Toutefois, étant légèrement en retard sur le planning de l'année, et sachant que nos algorithmes réalisaient à peu près ce qui était attendu d'un réseau de neurones : nous sommes passés à l'étape suivante du projet : l'apprentissage de la théorie des réseaux à convolution.

2 RÉSEAUX DE NEURONES À CONVOLUTIONS

Cette discipline des réseaux neuronaux n'a été dans un premier temps que survolée, et le but a surtout été d'en connaître les détails théoriques, et le fonctionnement. Des exemples extrêmement simplifiés ont été présentés par l'équipe de Eponge. La mise en pratique des réseaux à convolution a été possible grâce à l'étape suivante : le maniement de tensorflow.

2.1 Tensorflow

Nous avons donc, pour commencer, appliqué un tutoriel basique d'apprentissage sur MNIST, en CNN, fourni par tensorflow. Nous apprendrons plus tard que ce tutoriel utilisait des fonctions obsolètes, nous empêchant de travailler avec tensorboard, l'interface graphique de tensorflow. Toutefois, les résultats en terme d'efficacité de tensorflow étaient bien plus optimisés que nos réseaux neuronaux, et la grande diversité de paramètres nous a introduit à un concept qui ne nous avait pas intéressés jusque-là : le dropout.

Le dropout consiste à désactiver un certain pourcentage de neurones de façon aléatoire (p d'activation du neurone fixé en paramètre). Ce système d'apprentissage " partiel " permet en réalité aux neurones d'être moins dépendants des neurones précédents, et donc plus dépendants de l'entrée en elle-même, mais surtout permet d'éviter le surapprentissage. Toutefois, comme vu auparavant, la base MNIST de par sa taille et sa diversité empêche en elle-même le surapprentissage, ce concept sera donc à garder pour d'autres éventuels systèmes neuronaux.

Nous avons eu l'occasion de discuter avec un ancien élève de Supélec lors de l'une des nombreuses réunions de projet, qui travaillait également avec tensorflow sur de l'analyse de signaux. Cet élève nous a gracieusement fourni un code tensorflow, utilisant des fonctions actuellement à jour, et permettant la visualisation en direct de tensorboard et des différents graphiques liés à nos réseaux de neurones. Si la partie graphique de tensorboard est assez intuitive à prendre en main, la déconcertante complexité du code nécessaire à son bon fonctionnement a été une embûche à notre avancement.

Une fois l'étape des CNN terminée, les profs encadrants nous ont donné pour but de travailler sur le Q-learning : une théorie de l'apprentissage par machine tout à fait différente des perceptrons, avec des cas d'applications tout aussi différents.

SEMAINE DU 10 OCTOBRE La semaine a été consacrée à l'amélioration des premier résultats obtenus sur XOR, mais surtout sur MNIST.

L'utilisation de la tangente hyperbolique $f(x) = 1.7159 \times tanh(\frac{2x}{3})$ a permis d'améliorer les résultats, principalement en accélérant les calculs de propagation par rapport à $f(x) = \frac{1}{1+e^{-x}}$.

Les fonctions permettant de sauvegarder et de charger un perceptron après apprentissage dans un fichier ont été rajoutées. Mais la structure de donnée de l'autre groupe n'est pas forcément identique, il faut donc se mettre d'accord sur un format de fichier commun. Il sera ensuite possible de vérifier le comportement de propagation des deux perceptrons, qui devrait être identique.

SEMAINE DU 30 OCTOBRE L'équipe a continué à se renseigner sur les réseaux de neurones à convolution. L'accès aux machines virtuelles a permis de lancer des scripts plus longs sans handicaper les membres de l'équipe.

Sur un autre sujet, nous nous sommes penchés sur les différences (de l'ordre d'un facteur 100) entre notre taux d'apprentissage sur MNIST et celui de l'équipe Éponge.

3 APPRENTISSAGE PAR RENFORCEMENT

SEMAINE DU 11 DÉCEMBRE Loïc a présenté les premières bases théoriques du Q learning, ou apprentissage par renforcement. Ce genre d'algorithme de machine learning n'est pas utilisé pour des problèmes de classification, mais pour explorer les chemins d'un graphe inconnu, afin de trouver un chemin gagnant (pour une certaine fonction de succès) sans tomber sur un état de défaite.

REFERENCES