Rapport de Stage

Comparaison de méthodes d'apprentissage automatique

par Mathieu PONT

dans le cadre de la

Licence Informatique 3ème année à l'Université Paul Sabatier de Toulouse

effectué à

L'Institut de Recherche en Informatique de Toulouse (IRIT)

du

9 avril au 6 juillet 2018

Tuteurs de l'organisme d'accueil : M. Frederic MIGEON et M. Jerome MENGIN

Tuteur universitaire : M. Franck SILVESTRE

Introduction

Afin de clôturer ma formation dans le cadre de la Licence Informatique à l'Université Paul Sabatier de Toulouse, j'ai effectué un stage de 3 mois du 9 avril au 6 juillet 2018 à l'Institut de Recherche en Informatique de Toulouse (IRIT).

Cette expérience professionnelle a pour but de mettre en application des connaissances acquises tout au long des années de la Licence et de découvrir le monde professionnel.

Durant ce stage je devais comparer différentes méthodes d'apprentissage automatique en évaluant les performances d'un système multi-agents (déjà construit) et celles de réseaux de neurones (que je devais moi même mettre en œuvre).

Ce document retrace de façon synthétique toutes les activités que j'ai réalisé durant mon stage. Néanmoins ce rapport a été écrit peu après le milieu du stage et ne contiendra donc pas les derniers travaux effectués. Vous pourrez trouver les dernières mises à jour sur le dépôt github associé dont le lien est disponible en annexe.

Je présenterai dans un premier temps le lieu et l'équipe dans laquelle j'ai effectué mon stage. Dans un second temps, je parlerai en détail du sujet qui m'a été confié, des travaux effectués et de l'état d'avancement. Je finirai enfin par un bilan qui présentera à la fois sur le plan personnel et sur le plan professionnel les semaines passées au sein de l'IRIT.

J'aimerais remercier Frédéric MIGEON, Jérôme MENGIN, Bruno DATO et Franck SILVESTRE pour m'avoir accompagné durant ce stage, ainsi que toute l'équipe dans laquelle j'ai été durant ces 3 mois pour leur accueil chaleureux.

Je voudrais particulièrement remercier Frédéric MIGEON pour m'avoir permis de faire ce stage durant lequel j'ai appris énormément et où j'ai pu mettre en œuvre des notions qui m'intéressent grandement.

Présentation de l’organisme d’accueil

L’IRIT est une unité mixte de recherche organisée autour de différents thèmes.

- Analyse et synthèse de l'information
- Indexation et recherche d'informations
- **Intelligence Collective et Interaction**
- Raisonnement et décision
- Modélisation, algorithmes et calcul haute performance
- Architecture, systèmes et réseaux
- Sûreté de développement du logiciel

J’ai effectué mon stage au sein de l’équipe SMAC : Systèmes Multi-Agents Coopératifs. Ses recherches sont organisées dans le département « Intelligence Collective et Interaction ».

Un agent est une entité agissant de façon autonome dans un environnement et accomplissant des tâches en fonction du but qui lui est donné. Il peut percevoir (de manière limitée) son environnement et agir dans ce dernier.

Un système multi-agents (SMA) est composé d’un ensemble d’agents qui œuvrent tous à la tâche commune (mais ayant toujours chacun ses propres perceptions, objectifs etc.). Ces agents peuvent communiquer entre eux pour échanger des informations. Dans le cadre des systèmes multi-agents coopératifs, chaque agent possède un comportement basé sur une attitude ne nuisant pas aux autres agents.

Les SMA reposent en partie sur le principe de l’émergence résumé par « le tout est plus que la seule somme de ses parties ». C’est à dire que le comportement de la globalité d’un système peut être plus « complexe » que la somme des comportements de chaque constituant de ce système. On peut voir des propriétés émerger d’un tel système qui n’étaient pas forcément programmées ou prévues.

Présentation du sujet

L'apprentissage automatique est un champ d'étude de l'intelligence artificielle permettant à un système « d'apprendre » dans le sens où il peut améliorer de lui-même ses performances sur une tâche spécifique.

En 1959, Arthur Samuel définit ce terme par le « champ d'étude qui donne aux ordinateurs la capacité d'apprendre sans être explicitement programmés »[1]. En effet, dans la programmation « classique », nous devons définir de manière claire ce que le système doit faire pour chaque situation qu'il pourrait rencontrer. Avec l'apprentissage automatique, ce n'est plus le cas, c'est le système qui va trouver ce qu'il doit faire de lui même (d'où le terme « apprendre »).

Le système multi-agents auto-adaptatif AMOEBA a été développé dans le cadre de la thèse de Julien Nigon [Nigon 2017] pour réaliser de l'apprentissage automatique.

L'objectif du stage est de réaliser une étude comparative d'AMOEBA avec d'autres systèmes d'apprentissage dans le cadre de *benchmarks* tels que la plateforme MarioAI [Karkovskiy et al. 2012]. Je devais donc trouver des méthodes d'apprentissage performantes dans ce contexte, les mettre en œuvre et ensuite les comparer avec AMOEBA sur le jeu de Mario.

Il existe différentes catégories d'apprentissage, celle qui nous intéressera ici sera l'apprentissage par renforcement qui consiste à dire au système ce qu'il fait de bien ou de mal à l'aide de récompenses.

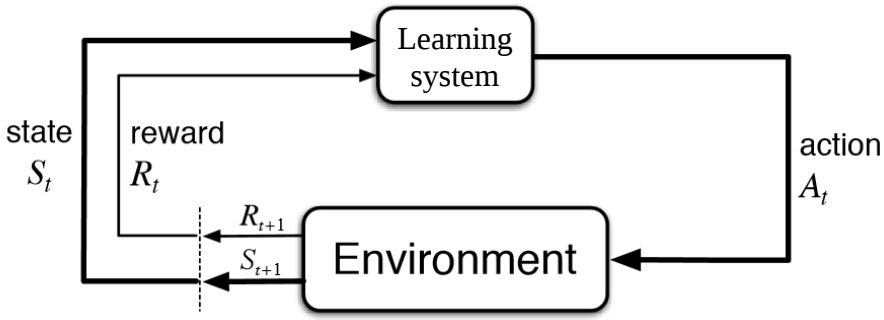


Figure 2.1 – Représentation d'un système apprenant et de son environnement dans le cadre d'un apprentissage par renforcement

Un système apprenant, se trouvant dans un environnement, effectue une action et observe l'effet de cette dernière sur l'environnement. Le système arrive dans un nouvel état (*state*) et peut recevoir une récompense (*reward*) dûe à l'action réalisée. L'objectif du système est de maximiser les récompenses reçues en choisissant les bonnes actions. En introduisant la notion de récompense, on définit ici la notion d'apprentissage par un problème d'optimisation de la récompense et de recherche des actions qui permettent cette optimisation.

1 https://toiledefond.net/le-machine-learning-quand-les-ordinateurs-predisent-lavenir/

Déroulement du stage

Nous devons d'abord nous demander, dans le cas du jeu de Mario, comment nous pouvons représenter un état qui correspond à ce que « voit » le système, ce sont les informations qu'il reçoit de l'environnement et à partir desquelles il fera des choix. Puis, comment représenter les récompenses qui permettront au système de savoir les actions qu'il doit privilégier et celles qu'il doit éviter dans un état donné.

3.1 Représentation d'un état et des récompenses du jeu de Mario

Nous avons plusieurs manières de représenter un état pour le jeu de Mario. L'une d'elle est de travailler directement sur l'image brute en pixels. Une autre consiste à utiliser une grille (fournie par le *benchmark* Mario, généralement de taille 19x19) qui décrit l'environnement autour du personnage comme montré en *annexe 3.1*. Chaque case contient une valeur pouvant représenter un obstacle, un ennemi, une pièce etc.

Lors des précédentes expérimentations avec AMOEBA une représentation plus simplifiée a été choisie consistant en 4 paramètres : la vitesse de Mario sur l'axe X et Y et la distance au premier obstacle à droite et en bas du personnage.

On peut utiliser comme récompense les points rapportés par les actions de Mario, cette valeur augmentera par exemple lorsqu'il trouve des pièces, des objets, finit le niveau et elle diminuera lorsqu'il rentre en collision avec des ennemis, perd le niveau etc.

Lors des précédentes expérimentations avec AMOEBA, une autre fonction de récompense a été choisie : Mario est récompensé de manière positive lorsqu'il avance vers la droite (la fin du niveau se situant à droite) et vers le haut (pour l'encourager à franchir des obstacles) et de manière négative lorsque c'est vers la gauche et le bas.

Dans un premier temps, nous avons choisi de comparer comme représentation d'état celle utilisant la grille et celle avec les 4 paramètres. En terme de fonction de récompense nous avons choisi de comparer celle calculée sur le score obtenu par Mario et celle utilisée par AMOEBA. Ce qui nous fait en tout 4 combinaisons de représentation d'état/récompense. Dans un second temps, nous pourrions aussi ajouter la combinaison des deux représentations d'états et des deux fonctions récompenses ce qui nous ferait en tout 9 combinaisons état/récompense.

3.2 Etat de l'art

J'ai commencé par faire un état de l'art sur les méthodes d'apprentissage par renforcement pertinentes dans notre contexte et trois principales en sont ressorties.

1 https://toiledefond.net/le-machine-learning-quand-les-ordinateurs-predisent-lavenir/

La première est le Deep Q Learning[1] [Mnih et al. 2013]. C'est un réseau de neurones à convolution[2] ayant pour objectif d'approximer une fonction nommée Q [Watkins 1989]. Cette dernière permet d'évaluer la qualité d'une action pour un état donné. Cette qualité représente « la récompense cumulée possible à la fin du jeu après avoir fait telle action dans tel état »[1]. Ainsi, le système peut choisir l'action qui a la plus grande qualité dans chaque état (c'est à dire l'action qui lui semble la meilleure dans telle situation). Le réseau peut utiliser la rétropropagation pour apprendre (c'est à dire trouver des bonnes valeurs pour les connexions entre les neurones qui l'amèneront à faire des prédictions de plus en plus correctes).

La deuxième est NEAT[3] (NeuroEvolution of Augmenting Topologies) [Stanley et al. 2002]. C'est un réseau de neurones utilisant les algorithmes génétiques pour faire croître son architecture afin de maximiser sa *fitness*. Cette valeur symbolise à quel point le réseau de neurones est performant pour la tâche en question.

La dernière méthode est un mélange des deux précédentes. Elle exploite le fonctionnement de NEAT pour trouver l'architecture adéquate et utilise le Q Learning avec la rétropropagation pour apprendre [Whiteson et al. 2006].

3.3 Mise en œuvre des systèmes

J'ai ensuite pris un Deep Q Network déjà implanté pour le jeu de Mario que j'ai largement modifié pour l'adapter à nos besoins. Premièrement, il fallait lier le système au *benchmark* de Mario qu'utilisait AMOEBA. Pour ce faire, j'ai réalisé une interface Java-Python (le *benchmark* étant en Java et le réseau de neurones en Python). Ensuite, il nous fallait utiliser une architecture différente pour les deux représentations d'état possibles, le détail des architectures choisies est disponible en *annexe 3.2*. Je me suis inspiré de différentes architectures pour réaliser les miennes que j'ai améliorées en essayant différents paramètres (nombre de couches, leur « type », nombre de neurones par couche etc.) et en les testant sur le jeu.

J'ai aussi rajouté une couche récurrente [Hausknecht et al. 2015]. Cette dernière permet de donner au système la notion de dépendance temporelle entre les données qu'il traite. Ainsi, le traitement d'un état est dépendant des états traités précédemment. Cela permet au réseau d'apprendre (ou de tirer des informations) sur une séquence d'état au lieu d'uniquement sur un seul à la fois.

Quant à AMOEBA, j'ai rajouté la représentation d'état de la grille et la fonction de récompense utilisant le score du jeu. La version de AMOEBA utilisée pour Mario était une ancienne version, j'ai modifié l'algorithme du choix des actions pour utiliser la nouvelle version d'AMOEBA. J'ai été obligé de le faire car l'ancienne version posait des problèmes avec la représentation d'état utilisant la grille.

Lors de la comparaison des deux systèmes, il fallait bien faire attention qu'ils soient homogénéisés, c'est à dire qu'ils utilisent la même représentation d'état, la même récompense, les mêmes hyper-paramètres (comme le ratio exploration/exploitation, c'est à dire le pourcentage de chance de réaliser une action aléatoire) etc.

1 https://ai.intel.com/demystifying-deep-reinforcement-learning/
2 http://cs231n.github.io/convolutional-networks/
3 http://gekkoquant.com/2016/03/13/evolving-neural-networks-through-augmenting-topologies-part-1-of-4/ *(et les parties 2, 3 et 4)*

Bilans

4.1 Etat d’avancement et perspectives

Pour réaliser la plupart de mes expérimentations j’ai utilisé le cluster de calcul OSIRIM[1]. J’ai du apprendre à l’utiliser afin d’exécuter à la fois les expériences de AMOEBA (utilisant des CPUs) et les expériences des réseaux de neurones (utilisant des GPUs).

Les principaux problèmes que j’ai rencontrés sont liés à l’interface Python-Java qui m’a pris du temps à faire fonctionner sur mon ordinateur et qui ne marche pas encore à ce jour sur OSIRIM. J’ai donc jusqu’à maintenant réalisé les expérimentations des réseaux de neurones sur mon ordinateur.

Les résultats avec AMOEBA et la représentation d’état des 4 paramètres sont disponibles mais pas encore ceux du réseau de neurones car je n’ai pas eu le temps de correctement ajuster ses paramètres. Je devrais terminer de l’ajuster d’ici les prochains jours.

Il reste encore à ce jour à attendre les résultats de AMOEBA pour la représentation d’état utilisant la grille (dont les algorithmes sont en exécution en ce moment sur OSIRIM). Je dois aussi lancer les expérimentations du réseau de neurones avec cette représentation d’état (sachant que j’ai déjà ajusté le réseau au travers de quelques expérimentations).

Ensuite, si j’ai le temps, je mettrai en application NEAT et si possible le système mélangeant les deux méthodes.

4.2 Bilan personnel

J’ai été extrêmement réjoui de faire ce stage. Il faut savoir que j’ai acquis en autodidacte toutes les connaissances que j’ai sur l’apprentissage automatique et ce qui l’entoure (surtout des réseaux de neurones). Bien que ce domaine soit lié à ma formation, nous n’avons aucun cours lors de la licence là-dessus (de tels cours sont plutôt dispensés en master).

Je suis donc fier d’avoir pu utiliser dans un cadre concret les connaissances obtenues sur mon temps libre.

Contrairement au cadre scolaire où nous sommes souvent encadrés pour nos projets et où on nous impose une démarche pour résoudre un problème, ici j'ai pu proposé la mienne. Une fois validée, cette démarche me laissait une certaine marge de manœuvre où je devais prendre des choix afin de réaliser au mieux les objectifs. Cela m'a conforté dans l'idée que j'en étais capable.

1 *Les expériences présentées dans ce rapport ont été (pour la plupart) réalisées en utilisant la plateforme OSIRIM qui est administrée par l'IRIT et soutenue par le CNRS, la région Midi-Pyrénées, le gouvernement français, et le FEDER (voir http://osirim.irit.fr/site/fr).*

4.3 Bilan professionnel

J’ai appris énormément lors de ce stage dans la conception de réseau de neurones et leur « tuning » (c’est à dire trouver de bons paramètres permettant au réseau d’être performant pour la tâche en question). J’ai fait des erreurs qui m’ont beaucoup appris et m’ont améliorés (comme par exemple le fait d’avoir appris à utiliser OSIRIM trop tard dans le stage ou encore de n’avoir pas eu connaissance de l’impact de certains paramètres sur le réseau de neurones).

Mon objectif à court terme est d’intégrer un master dans le domaine de l’intelligence artificielle et une telle expérience est plus que bénéfique pour cette poursuite d’étude.

Pour ce qui est du long terme, j’aimerais être chercheur dans ce domaine et ces 3 mois m’ont encore plus motivés dans la réalisation de ce but qui nécessiterait de faire un doctorat.

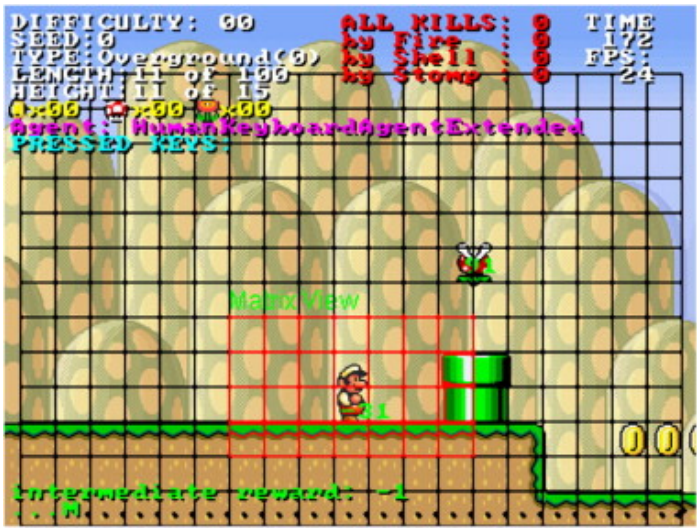
De plus, le fait d’avoir été dans l’équipe SMAC m’a fait beaucoup m’intéresser aux systèmes multi-agents alors que jusqu’à aujourd’hui, je ne m’intéressais qu’aux réseaux de neurones (pour ce qui est du domaine de l’intelligence artificielle).

Annexes

Dépot github : *https://github.com/MatPont/MarioBros_MachineLearning*

3 Déroulement du stage

- Annexe 3.1 – Grille d'observation de Mario



- Annexe 3.2 – Architectures des réseaux de neurones (avec le Q Learning)

Architecture (actuelle) pour la représentation d'état utilisant la grille :

Couche de convolution 1	16 filtres de taille 9*9 (stride=4)
Couche de convolution 2	32 filtres de taille 5*5 (stride=2)
Couche de convolution 3	64 filtres de taille 3*3 (stride=1)
Couche LSTM*	512 neurones
Couche de sortie	14 neurones (pour les 14 actions possibles)

Architecture (actuelle) pour la représentation d'état utilisant les 4 paramètres :

Couche d'entrée	4 neurones
Couche LSTM*	8 neurones
Couche de sortie	14 neurones

*une couche LSTM est une couche récurrente dont une explication est disponible ici :

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Bibliographie

Mario

[Karkovskiy et al. 2012] Sergey Karakovskiy and Julian Togelius. The Mario AI Benchmark and Competitions. 2012.

[Curran et al. 2015] William Curran, Tim Brys, Matthew E. Taylor and William D. Smart. Using PCA to Efficiently Represent State Spaces. 2015.

AMOEBA

[Nigon 2017] Julien Nigon. Apprentissage artificiel adapté aux systèmes complexes par auto-organisation coopérative de systèmes multi-agents. 2017.

Deep Reinforcement Learning (Deep Q Learning)

[Watkins 1989] C. J. Watkins. Learning from delayed rewards. 1989.

[Liao et al. 2012] Y. Liao, K. Yi and Z. Yang. Final Report Reinforcement Learning to Play Mario. 2012.

[Mnih et al. 2013] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra and M. Ridmiller. Playing Atari with Deep Reinforcement Learning. 2013.

[Mnih et al. 2015] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg and D. Hassabis. Human-level control through deep reinforcement learning. 2015.

[Hasselt et al. 2015] H. Hasselt, A. Guze and D. Silver. Deep Reinforcement Learning with Double Q-learning. 2015.

[Hausknecht et al. 2015] Matthew Hausknecht and Peter Stone. Deep recurrent Q-learning for partially observable MDPs. 2015.

[Klein 2016] Sean Klein. Final Report Deep Q-Learning to Play Mario. 2016.

[Schaul et al. 2016] Tom Schaul, John Quan, Ioannis Antonoglou and David Silver. Prioritized Experience Replay. 2016.

[Lillicrap et al. 2016] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver and D. Wierstra. Continuous Control With Deep Reinforcement Learning. 2016.

[Mnih et al. 2016] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Tim Harley, Timothy P. Lillicrap, David Silver and Koray Kavukcuoglu. Asynchronous Methods for Deep Reinforcement Learning. 2016.

[Wolfshaar 2017] Jos van de Wolfshaar. Deep Reinforcement Learning of Video Games. 2017.

[Andersen 2018] P. Andersen. Deep Reinforcement Learning using Capsules in Advanced Game Environments. 2018.

Evolutionnary Algorithm (NEAT)

[Montana et al. 1989] David J. Montana and Lawrence Davis. Training Feedforward Neural Networks Using Genetic Algorithms. 1989.

[Stanley et al. 2002] K. O. Stanley and R. Miikkulainen. Evolving Neural Networks through Augmenting Topologies. 2002.

[Stanley et al. 2003] Kenneth O.Stanley, Bobby D. Bryant, Risto Miikkulainen. Evolving Adaptive Neural Networks with and without Adaptive Synapses. 2003.

[Hausknecht et al. 2013] M. Hausknecht, J. Lehman, R. Miikkulainen and P. Stone. A Neuroevolution Approach to General Atari Game Playing. 2013.

[Miikkulainen et al. 2017] Risto Miikkulainen, Jason Liang, Elliot Meyerson, Aditya Rawal, Dan Fink, Olivier Francon, Bala Raju, Hormoz Shahrzad, Arshak Navruzyan, Nigel Duffy and Babak Hodjat. Evolving Deep Neural Networks. 2017.

NEAT and Backpropagation (with Q Learning)

[Whiteson et al. 2006] Shimon Whiteson and Peter Stone. Evolutionary Function Approximation for Reinforcement Learning. 2006.

[Whiteson 2010] Shimon Whiteson. Adaptive Representations for Reinforcement Learning. 2010.

[Land et al. 2017] Julien van der Land, Thomas Nijman, Stephan Boomker and Jasper de Boer. Playing a volleyball game with reinforcement learning. 2017.