TP 4 : Projet de recherche

Application de la neuro-évolution sur des environnements de OpenAi Gym

Bernard Meunier, Alexia Reynaud, Olivier Naud-Dulude, Charles-Auguste Marois

Bernard Meunier : Génie informatique à Polytechnique Montréal, matricule 1878557
bernard.meunier@polymtl.ca

Alexia Reynaud : Génie électrique à Polytechnique Montréal, matricule 1845955
alexia.reynaud@polymtl.ca

Olivier Naud-Dulude : Génie logiciel à Polytechnique Montréal, matricule 1881393
olivier.naud-dulude@polymtl.ca

Charles-Auguste Marois : Génie informatique à Polytechnique Montréal, matricule 1850196
charles.marois@polymtl.ca

Abstract

Les réseaux de neurones profonds sont généralement entrainés à partir d'algorithmes d'apprentissage par descente du gradient tels que *Q-learning* et Policy Gradient. Toutefois, les stratégies de neuro-évolution basées sur les algorithmes génétiques permettent le même genre d'entraînement [Mnih et al. 2018]. Nous avons donc voulu tester différents environnements pour confirmer qu'il est possible d'utiliser des algorithmes génétiques pour entraîner efficacement des réseaux de neurones profonds.

1 Introduction

Lors d'un problème d'apprentissage par renforcement (Reinforcement Learning ou RL), le réseau de neurones cherche généralement à maximiser un gain cumulatif (total ou actualisé) sans supervision sur la façon de maximiser ce dit gain. Jusqu'à présent, trois familles d'algorithmes se sont avérées efficaces pour la résolution de problèmes de RL, soit les méthodes Q-Learning (DQN) [Mnih et al., 2015], les méthodes de Policy gradient (A3C, TRPO, PPO) [Sehnke et al., 2010] et plus récemment les stratégies neuro-évolutives [Salimans et al., 2017].

Les algorithmes de Q-learning estiment la fonction Q optimale d'un réseau de neurones profond en produisant une politique qui, pour un état quelconque, choisit l'action qui maximise la Q-value. En d'autres mots, le réseau qui joue le rôle de la fonction Q prend à son entrée les états d'un agent et retourne en sortie, via la fonction d'activation softmax, une action discrète.

Les méthodes de Policy gradient apprennent directement les paramètres d'une politique de réseau de neurones profonds qui génère la probabilité de prendre chaque action dans chaque état. Elles visent à éviter les intégrales trop laborieuses à calculer en utilisant le gradient d'une politique déterministe au lieu de celui d'une politique stochastique. Ainsi, on peut améliorer la performance et l'efficacité dans l'estimation du gradient par rapport au Q-learning.

Enfin, une équipe d'OpenAi a récemment mis au point une version simplifiée de Natural Evolution Strategies qui apprend la moyenne d'une distribution de paramètres, mais non sa variance. Ils ont constaté que cet algorithme est compétitif avec les deux autres familles d'algorithmes sur des problèmes de RL difficiles avec des temps d'entraînement beaucoup plus rapides (lorsque de nombreux processeurs sont disponibles) grâce à une meilleure parallélisation [Salimans et al., 2017].

2 La neuro-évolution

L'approche de l'intelligence artificielle du style neuro-évolution est une approche inspirée de l'évolution biologique du système nerveux des êtres vivants. Le but est de faire évoluer un réseau neuronal capable de prendre des décisions selon l'état de la situation dans laquelle il se trouve.

Pour ce faire, il faut avoir un environnement qui est capable d'être observé, soit par des données quantitatives sur différents éléments, soit simplement par des images ou des vidéos. L'environnement doit aussi être capable d'accepter une décision et de déterminer si cette dernière est une bonne décision ou non dans le but d'atteindre un objectif. C'est grâce à ces stimuli qu'il est possible de faire évoluer un réseau neuronal qui permettra d'atteindre l'objectif voulu de la façon la plus efficace possible.

C'est l'aspect d'évolution de la neuro-évolution qui est le plus représentatif de l'évolution biologique. En effet, on commence par générer une population d'agents capables de prendre des décisions par rapport aux observations sur l'environnement. Cette décision est prise par un réseau neuronal qui est généré de façon aléatoire, donc les décisions prises le sont toutes autant. Cette population d'agents est notre première génération d'agents. On teste alors chacun de ces agents par rapport à notre environnement et on attribue à chacun d'eux un score selon leur performance à atteindre l'objectif voulu.

Ensuite, il faut faire évoluer la population d'agents. Pour cela, nous commençons par sélectionner

certains de ces agents qui serviront de *parents* à la prochaine génération. Le procédé de cette sélection peut être fait de différentes façons et peut avoir différents impacts sur le processus d'évolution. Plus on sélectionne uniquement les meilleurs agents, plus on risque de perdre de la diversité génétique et plus on prend le risque d'atteindre un optimum local qui n'est pas le meilleur. Cependant, plus on garde des agents de façon aléatoire, moins rapidement l'algorithme convergera vers un optimum.

Ensuite, il faut générer une nouvelle génération d'agents avec les parents sélectionnés. Cependant, les nouveaux agents créés doivent être une évolution de leurs parents pour pouvoir avoir la chance d'être meilleurs. C'est pour cela que la création des agents *enfants* se fait par le croisement de parents. Ce croisement peut être fait de plusieurs façons. Le but est de construite un agent avec son réseau neuronal fait à partir de celui des parents. Par exemple, on peut sélectionner au hasard la provenance de chacun des poids du nouveau réseau pour en faire un nouveau unique. Par la suite, pour s'assurer de faire apparaitre du matériel génétique nouveau dans la banque de matériel génétique que nous avons, il faut appliquer des mutations sur la nouvelle génération. Tous les nouveaux agents créés par croisement sont passés dans un algorithme de mutation. Par exemple, on peut itérer sur chacun des poids du réseau et selon une probabilité, le remplacer par un nouveau généré aléatoirement.

Une fois cette nouvelle génération créée, on recommence le processus de test sur chacun des agents et ensuite la sélection des *parents* en créant une nouvelle génération. Ce cycle peut se réaliser un nombre prédéterminé de fois ou peut être arrêté plus rapidement si certains critères de performance sont atteints. On garde alors le meilleur agent généré et on lui fait procéder une évaluation complète de ses performances pour confirmer ses capacités.

3 L'environnement utilisé

Pour l'expérimentation de notre projet, nous avons décidé de créer un algorithme de neuro-évolution pour résoudre les différents environnements fournis par la librairie *Gym* de la compagnie *OpenAI*.[1] Cette librairie est une boite à outils permettant de développer différents algorithmes d'apprentissage par renforcement. Elle comprend une collection d'environnements de tests ayant tous une interface d'utilisation semblable pour expérimenter avec différents algorithmes. Le but de l'expérimentation est de développer l'algorithme générique de neuro-évolution pour résoudre l'environnement appelé « CartPole-v1 ». Ensuite, nous appliquons ce même algorithme à d'autres environnements avec comme seule modification les hyperparamètres de l'algorithme.

La construction de notre algorithme de neuro-évolution a été effectuée en deux étapes. Premièrement, nous voulions faire l'algorithme sans se baser sur la littérature existante et seulement à partir de nos propres raisonnements. Une fois l'algorithme fonctionnel, nous avons amélioré incrémentalement notre algorithme à partir de différentes méthodes existantes dans la littérature et avons comparé les résultats obtenus.

3.1 Construction de l'algorithme

Le réseau neuronal

Nous avons choisi de ne pas utiliser de librairie de réseau neuronal pour avoir un contrôle complet sur l'acheminement et l'évolution des données. Nous avons commencé par la création de la classe du réseau neuronal. L'implémentation effectuée est flexible par rapport au nombre de couches cachées et au nombre de neurones dans chacune des couches. Un biais est initialisé pour chacune des couches.

La fonction d'activation choisie sur les couches cachées est la fonction de rectification linéaire ReLU. Nous avons choisi cette fonction d'activation pour sa simplicité et ses bonnes performances. Pour la couche de sortie du réseau, nous avons choisi la

[1] Source : https://gym.openai.com/

fonction d'activation exponentielle normalisée Softmax. Cette fonction a été choisie puisqu'elle est efficace pour les algorithmes de classification. Effectivement, dans les environnements du *Gym*, une seule action peut être prise à la fois. Il s'agit alors d'effectuer la classification de la meilleure action possible à chaque décision.

C'est dans la classe de réseau neuronal que nous avons défini notre algorithme de croisement et de mutation. Nous avons utilisé deux algorithmes de croisement. Premièrement, nous avons utilisé une solution maison où l'on parcourait itérativement tous les poids des deux réseaux neuronaux parents. Pour chacun des poids, on décidait au hasard de quels parents faire le poids correspondant du nouveau réseau enfant. Nous référerons à cet algorithme par la technique poids par poids. Deuxièmement, nous avons utilisé la technique de croisement par optimisation combinatoire trouvée dans l'article *« An alternative approach for neural network evolution with a genetic algorithm: Crossover by combinatorial optimization »* [Garcias-Pedrajas et al., 2005]. Cette approche consiste à garder groupés ensemble les poids influençant chacun des nœuds et à créer le réseau neuronal enfant en choisissant aléatoirement le parent pour chacun de ces groupes de nœuds.

Les agents

La classe agent a pour but d'être une interface entre l'environnement de test et le réseau neuronal. Chaque agent garde en mémoire son *pointage* qui est l'indice de sa performance à accomplir la tâche donnée et un objet de la classe réseau neuronal.

Dans son constructeur, la classe agent reçoit de l'information par rapport à l'environnement utilisé. Celle-ci s'en sert pour générer un réseau neuronal ayant le nombre de nœuds d'entrées et de sortie adaptées à l'environnement. Cela permet d'utiliser la classe agent et réseau neuronal sans modification sur plusieurs environnements de test du *Gym*.

La sélection des parents

La technique utilisée pour la sélection des parents après le test de chaque génération peut avoir un très grand impact sur le résultat final de l'algorithme. Notre algorithme maison était de classer par ordre de performance tous les agents de la population observés et de ne garder que les meilleurs agents selon un nombre fixe prédéterminé. Nous nommerons cette approche la sélection par classement pour y référer. Par la suite, nous avons changé cet algorithme par un qui est basé avec la conclusion de l'article *« Parent Selection Operators for Genetic Algorithms* » [Jebari et al., 2013]. Cet article propose plusieurs types d'algorithmes de sélection et indique leurs forces et faiblesses respectives. La conclusion de l'article est que l'algorithme devrait attribuer une probabilité de sélection d'un agent autant basée sur son pointage par rapport au reste du groupe que sur la position dans un classement de performance dans la population. Nous avons donc utilisé un algorithme de type « The Tournement Selection » ou TOS. Cet algorithme consiste à sélectionner le meilleur agent dans un sous-ensemble de la population déterminé au hasard. On recommence ces tournois pour le nombre de parents désirés. Ensuite nous nous assurons de sélectionner le meilleur agent de la population et nous le rajoutons à la liste de parents. Nous avons choisi cet algorithme pour sa rapidité de convergence dans de petites populations ainsi que sa faculté de garder une grande diversité génétique entre les agents.

L'entraînement

Le fonctionnement de notre algorithme est simple. Premièrement, une population d'agents est générée au hasard contenant des réseaux neuronaux. Les hyperparamètres d'entrées et de sorties de ces réseaux sont déterminés par l'environnement et les couches cachées par l'utilisateur. Chaque agent de la population est alors testé un nombre de fois prédéterminé et son pointage est moyenné par rapport au nombre de tests. Cela permet d'avoir des algorithmes capables de s'adapter aux variations aléatoires de l'environnement. Le pointage d'un agent pour un test est déterminé par la

sommation des récompenses obtenues après chaque action.

Grâce aux différents processus décrits plus haut, des parents sont alors sélectionnés et une nouvelle population créée. Ce cycle recommence pour un nombre maximal prédéterminé de générations. Après toutes ces générations, le meilleur agent est sélectionné pour l'évaluation.

Plusieurs environnements de la librairie *Gym* ont une limite de pointage maximal qu'un agent peut atteindre. Une fois cette limite atteinte par un agent, celui-ci est sélectionné pour l'évaluation et l'algorithme est arrêté de façon hâtive.

L'évaluation

Les différents environnements de la librairie Gym viennent avec un objectif. Que ce soit d'atteindre un but le plus rapidement possible ou de faire *survivre* un agent le plus longtemps possible, un pointage est attribué à l'agent et celui-ci peut être comparé à un pointage cible donné par l'environnement. L'évaluation consiste donc à tester plusieurs centaines de fois l'agent et de moyenner ses résultats. Si cette moyenne est au-dessus du seuil prédéterminé par les développeurs de l'environnement, l'expérience est considérée comme un succès.

4 Expérimentations

Nos expérimentations se sont très bien déroulées. Comme précisé ultérieurement, nous avons optimisé notre algorithme autour de l'environnement « CartPole-v1 ». Cet environnement consiste en un wagon sur rail et un poteau connecté à l'aide d'une charnière libre.

Les actions possibles dans l'environnement sont de déplacer le wagon vers la droite et la gauche et l'objectif est de maintenir le poteau en équilibre le plus longtemps possible. À chaque trame d'image, une action doit être choisie et la récompense est d'un point. Si le poteau perd l'équilibre, c'est-à-dire s'il tombe de ±12° de chaque côté, le test est terminé. Cependant, après 500 trames d'images, si le poteau n'est toujours pas tombé, le test est un succès et obtient le pointage maximal de 500.

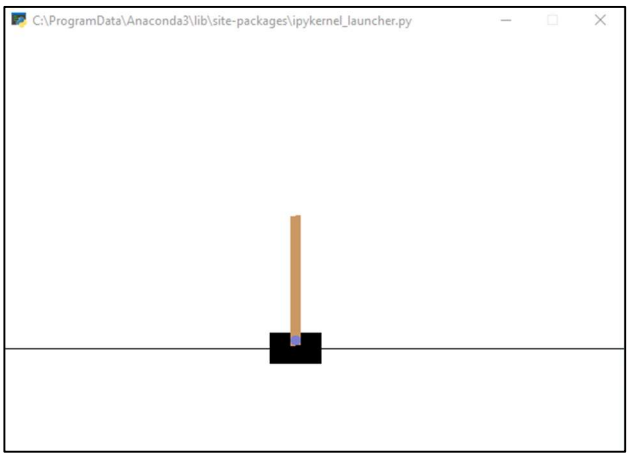


Image 1 : Représentation de l'environnement de CartPole-V1. Les flèches rouges montrent les actions possibles.

Pour tester la performance de notre algorithme, nous avons choisi d'utiliser une population de 100 agents et de les tester 15 fois par génération pour un maximum de 15 générations. À chaque nouvelle génération, 10 agents sont sélectionnés pour être les parents de la prochaine génération. Lorsqu'un agent atteignait le pointage maximal, on notait le nombre de générations utilisées. Le tout était recommencé 50 fois et moyenné. Ici, nous considérons que plus rapidement le pointage maximal est atteint, meilleur est l'algorithme.

Nous avons exécuté quatre fois le test décrit plus haut. Premièrement nous avons testé notre algorithme de sélection par classement et notre algorithme de croisement poids par poids. Deuxièmement, nous avons utilisé l'algorithme de sélection de parents TOS et celui de croisement poids par poids. Troisièmement, nous avons testé avec notre algorithme de sélection par classement ainsi que l'approche de croisement par optimisation combinatoire. Finalement, nous avons utilisé l'algorithme de sélection TOS et l'approche par optimisation combinatoire. La table suivante montre le

nombre moyen de générations pour obtenir un agent ayant le pointage maximal.

	Sélection par classement	TOS
Poids par poids	8.575	9.41
Optimisation combinatoire	7.56	9.81

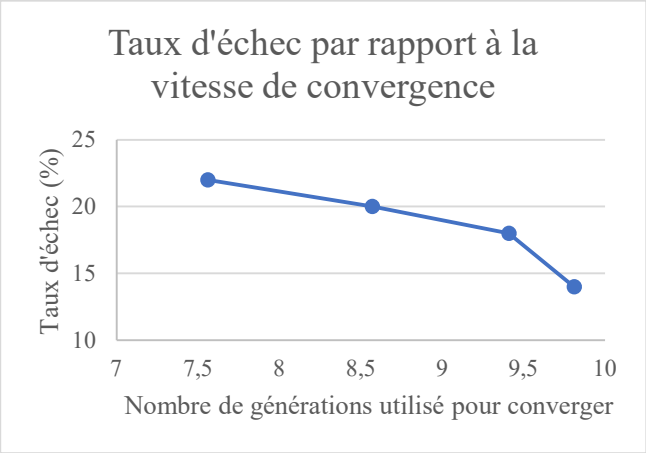
Table 1 : Nombre de générations pour obtenir un score parfait selon différents algorithmes.

Nous avons aussi enregistré le taux d'échec des algorithmes. Nous considérons qu'un échec arrive si, après 15 générations, l'algorithme n'a pas réussi à atteindre une convergence. Voici la table de nos observations.

	Sélection par classement	TOS
Poids par poids	20%	18%
Optimisation combinatoire	22%	14%

Table 2: Pourcentage d'échec à la convergence vers une solution après 15 générations.

Nous trouvons ces résultats très intéressants. Nous pouvons voir que l'algorithme le plus rapide a été la combinaison de la sélection par classement et de l'optimisation combinatoire. Cependant cet algorithme a aussi été celui qui a le plus grand taux d'échec parmi toutes les combinaisons. En regardant le reste des résultats, nous pouvons facilement voir que plus la convergence moyenne est rapide, plus le taux d'échec est élevé. Cette relation est presque linéaire, comme on peut le voir sur le graphique suivant.



Graphique 1 : Taux d'échec par rapport à la vitesse de convergence.

Le résultat obtenu confirme nos hypothèses par rapport à l'algorithme TOS. Cet algorithme propose un entre-deux entre la perte de matériel génétique dans la population et la sélection des meilleurs agents. C'est pour cela que l'on aperçoit des taux d'échec de convergence plus bas, mais aussi une vitesse de convergence plus basse. Du côté de l'algorithme de croisement par optimisation combinatoire, son impact sur les résultats semble dépendre de notre algorithme de sélection. Dû au fait qu'il garde groupé les poids de chaque nœud, nous en concluons qu'il est mieux adapté dans les populations ayant une plus grande variété génétique.

5 Autres environnements

Comme prévu, nous avons utilisé notre algorithme sur d'autres environnements de test de la librairie Gym. Pour avoir un taux de succès le plus élevé possible, nous avons utilisé l'algorithme de sélection TOS et le croisement par optimisation combinatoire.

Premièrement, nous avons essayé l'environnement appelé « Acrobot-v1 ». Le défi consiste à faire monter une pendule à deux segments au-dessus d'une ligne fixe le plus rapidement possible en appliquant des forces sur les charnières.

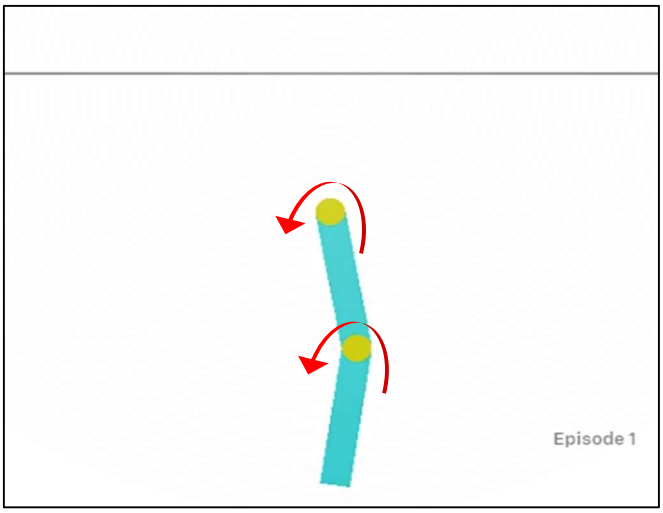


Image 2 : Représentation de l'environnement de Acrobot-V1. Les flèches rouges montrent les actions possibles.

Notre algorithme a été instantanément capable de résoudre le défi, même avec les différences de fonctionnement du pointage. Effectivement, contrairement à un environnement où chaque trame d'image augmente le pointage de l'agent, chaque trame diminue le pointage de l'agent tant que le succès n'est pas obtenu. Cela insinue qu'un agent doit être capable de réussir le défi pour pouvoir se démarquer des autres.

Le prochain environnement essayé s'appelle « MountainCar-v0 ». Il s'agit d'une voiture coincée entre deux montagnes. Elle doit atteindre un drapeau sur le montage de droite le plus rapidement possible, mais son moteur n'est pas assez puissant. La solution optimale consiste à reculer sur la montagne de gauche pour pouvoir se donner plus de vitesse pour gravir la montagne. Les actions possibles sont de pousser la voiture vers la droite, vers la gauche ou ne rien faire.

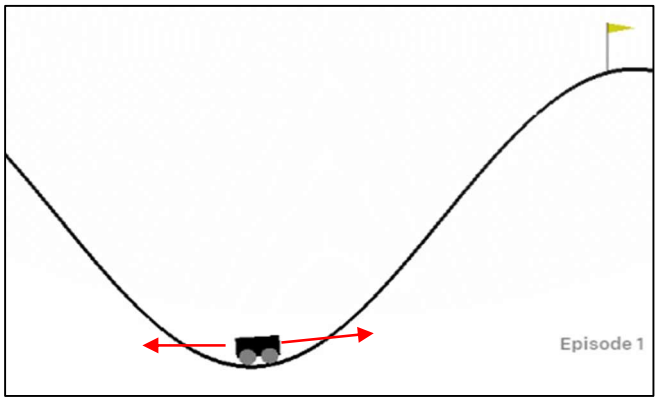


Image 3 : Représentation de l'environnement de MountainCar-v0. Les flèches rouges montrent les actions possibles.

Malheureusement, notre algorithme n'a pas été capable de résoudre ce défi. Le pointage d'un agent est diminué de 1 à chaque trame d'image. Pour pouvoir se démarquer, un agent doit se rendre jusqu'au drapeau. Malheureusement, aucun de nos agents ne se rendait au drapeau donc tous les agents restaient avec un pointage identique, même si certains étaient plus proche de la solution que d'autres. Il était alors impossible d'amorcer un processus d'apprentissage de génération à génération. Nous pensons que différents algorithmes d'apprentissage par renforcement seraient plus appropriés pour la situation. Peut-être qu'un algorithme de Q-Learning pourrait fonctionner.

6 Conclusion

Notre projet de découverte et d'exploration de l'apprentissage machine par neuro-évolution s'est très bien passé. Nous avons compris que plus un algorithme est rapide, plus il a de chance de converger vers un optimum local. Il faut donc trouver un balancement idéal entre la vitesse et la diversité génétique.

Nous avons aussi compris que certains types d'algorithmes sont plus utiles que d'autres en fonction de la situation. Même avec la construction d'un algorithme pouvant s'adapter à l'environnement, il ne sera pas nécessairement capable de résoudre le défi.

Dans de futurs travaux, il serait intéressant de faire varier plus que les poids du réseau neuronal, mais

aussi le nombre de nœuds par couche et le nombre de couches. Cela permettra d'obtenir une encore plus grande variété génétique dans la population.

7 Lien vers le GitHub

https://github.com/beurnii/INF8225

8 Références

[Garcias-Pedrajas et al., 2005] Garcias-Pedrajas N., Ortiz-Boyer D., and Hervas-Martinez C. *An alternative approach for neural network evolution with a genetic algorithm: Crossover by combinatorial optimization*. Department of Computing and Numerical Analysis, University of Cordoba. Cordoba, Span, 2005.

[Jebari et al., 2013] Jebari K. and Madiafi, Abdelaziz Elmoujahid M. *Parent Selection Operators for Genetic Algorithms.* LCS Laboratory, Faculty of Sciences, Mohammed V-Agdal University. Rabat, Morocco, 2013.

[Mnih et al., 2015] Mnih V., Kavukcuoglu K., Silver D., Rusu A. A., Veness J., Bellemare M. G., Graves A., Riedmiller M., Fidjeland A. K., Ostrovski G., and others. *Human-level control through deep reinforcement learning.* Nature, 2015.

[Salimans et al., 2017] Salimans T., Ho J., Chen X., and Sutskever I. *Evolution strategies as a scalable alternative to reinforcement learning*. arXiv preprint arXiv:1703.03864, 2017.

[Sehnke et al., 2010] Sehnke F., Osendorfer C., R¨uckstieß T., Graves A., Peters J., and Schmidhuber J. *Parameter-exploring policy gradients*. Neural Networks, 2010.

[Such et al., 2018] Such P., Madhavan V., Conti E., Lehman J., Stanley K., Clune J. *Deep Neuroevolution: Genetic Algorithms are a Competitive Alternative for Training Deep Neural Networks for Reinforced Learning*, 2018.