Sorbonne Université
Master ANDROIDE

Apprentissage par renforcement profond : analyse détaillée de SVPG

UE de projet M1

Julien CANITROT, Jules DUBREUIL, Tan Khiem HUYNH, Nikola KOSTADINOVIC

2022

Table des matières

La majuscule est sur le A ;)

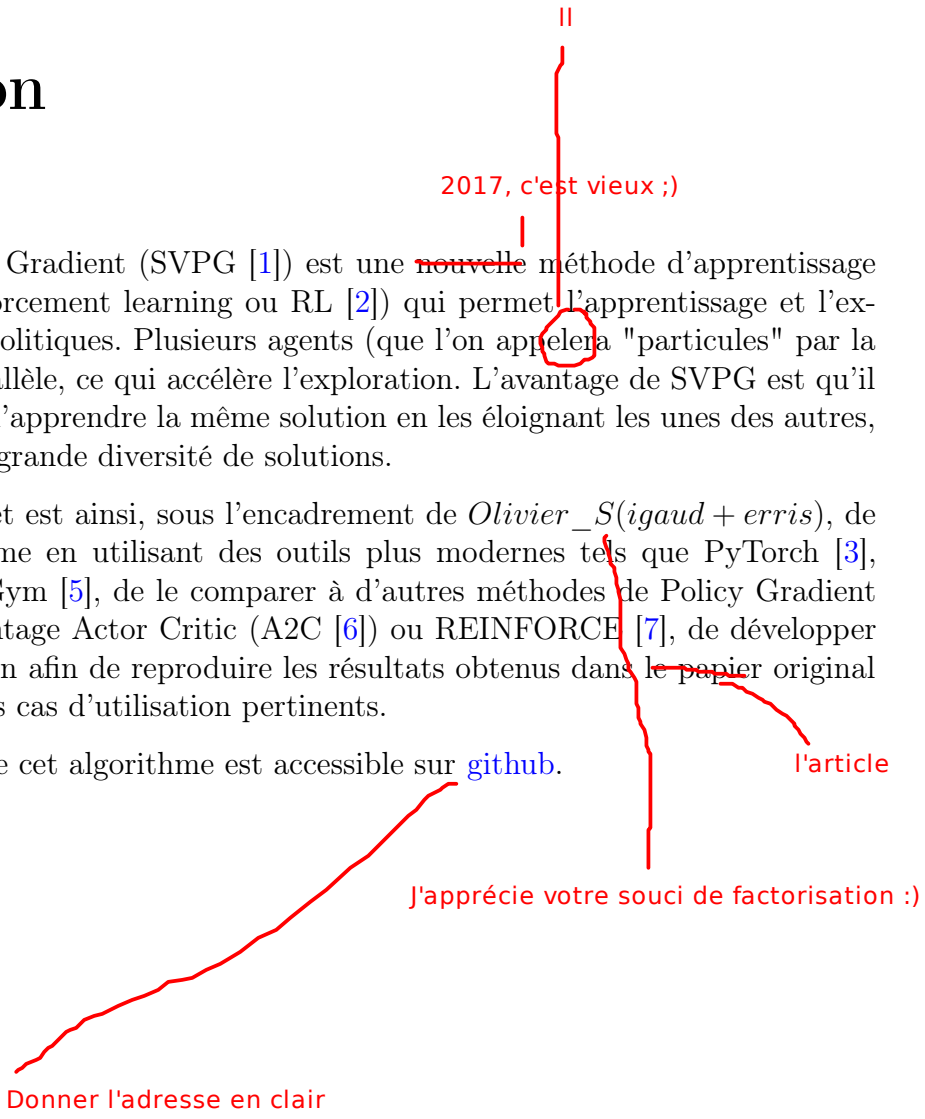
Chapitre 1

Introduction

Stein Variational Policy Gradient (SVPG [1]) est une ~~nouvelle~~ méthode d'apprentissage par renforcement (reinforcement learning ou RL [2]) qui permet l'apprentissage et l'exploitation de plusieurs politiques. Plusieurs agents (que l'on appelera "particules" par la suite) travaillent en parallèle, ce qui accélère l'exploration. L'avantage de SVPG est qu'il empêche ces particules d'apprendre la même solution en les éloignant les unes des autres, ce qui favorise une plus grande diversité de solutions.

L'objectif de notre projet est ainsi, sous l'encadrement de $Olivier\_S(igaud + erris)$, de moderniser cet algorithme en utilisant des outils plus modernes tels que PyTorch [3], SaLiNa [4] et OpenIA Gym [5], de le comparer à d'autres méthodes de Policy Gradient classiques comme Advantage Actor Critic (A2C [6]) ou REINFORCE [7], de développer des outils de visualisation afin de reproduire les résultats obtenus dans le papier original et de mettre en avant les cas d'utilisation pertinents.

Notre implémentation de cet algorithme est accessible sur github.



Chapitre 2

État de l'art

Nous présentons ici les concepts nécessaires à la compréhension de notre projet.

2.1 Méthodes de Policy Gradient

Les méthodes de Policy Gradient, comme A2C ou REINFORCE, sont un type de techniques d'apprentissage par renforcement qui reposent sur l'optimisation de politiques paramétrées par rapport à la valeur de récompense attendue par descente de gradient. Cependant, ces méthodes souffrent généralement de variance élevée, de convergence lente ou d'exploration inefficace (Liu & Wang, 2017). L'algorithme SVPG tente de répondre à ces problèmes en s'appliquant par dessus ces méthodes. Il appartient à la catégorie des méthodes d'apprentissage par renforcement distribué (paper distributed RL [??]), une sous-partie du RL dans laquelle plusieurs particules travaillent en parallèle pour faciliter la parallélisation des calculs [??] et l'exploration des paramètres.

2.2 Stein Variational Gradient Descent (SVGD)

En RL il est nécessaire de pouvoir approximer une distribution de probabilité sur les différents actions possibles afin d'obtenir une idée sur les issues probables de nos politiques. Les méthodes usuelles pour ces approximations sont l'inférence variationnelle (Variational Inference ou VI [8]) et Markov Chain Monte Carlo Sampling (MCMC [??]). SVPG lui, repose sur Stein Variational Gradient Descent (SVGD [9]), un algorithme d'inférence bayésienne des mêmes auteurs. Il combine les avantages de MCMC qui ne confine pas l'approximation dans une famille paramétrique, et de VI qui converge rapidement grâce à des mises à jour déterministes qui utilisent le gradient.

SVGD trouve un compromis entre une force *attractive*, appliquée par le gradient sur les différentes particules, et une force *répulsive* afin de garder une part d'exploration entre ces mêmes particules. Pour la répulsion, SVGD se base sur la méthode de Stein [??], méthode utilisée en théorie des probabilités afin d'obtenir des bornes sur des distances entre deux distributions selon une certaine divergence. À partir de cette méthode (plus particulièrement de l'*identité de Stein*) on peut caractériser l'écart $\mathbb{S}(p, q)$ entre différentes distributions p et q dites *smooth* (dérivables) [??]. Avec cet écart, nous pouvons faire réagir ~~nos~~ particules les unes par rapport aux autres.

les (ne soyez pas possessifs ;))

Dans les faits, ce calcul nécessite une optimisation variationnelle très complexe le rendant quasiment insoluble pour nos ordinateurs. C'est pourquoi on définit un Kernelized Stein Discrepancy (KSD [10]) pour caractériser cet écart, que l'on applique par dessus des méthodes de Policy Gradient pré-existantes (A2C, REINFORCE).

2.3 Stein Variational Policy Gradient (SVPG)

SVPG a déjà été mis en application dans ce contexte [1]. Il a été identifié dans cet article les différents avantages de la méthode SVPG par rapport aux méthodes classiques d'apprentissage par renforcement. On comprend notamment que SVPG va explorer l'espace des paramètres de manière plus intelligente en exploitant les zones les plus fructueuses grâce à l'apprentissage commun des particules. Ce qui lui permet d'apprendre des politiques diverses et performantes. Ses performances sont surtout remarquables sur des environnements complexes comme CartPole Swing-Up et Double Pendulum.

Afin de gérer l'impact de la force répulsive, les auteurs introduisent un facteur α devant le terme de répulsion, qui permet de choisir entre une politique plutôt exploratrice ou exploitatrice. Le papier insiste sur l'intérêt de tester différentes méthodes de sélection de α pour rendre l'algorithme plus adaptatif lors de son apprentissage.

Dans le papier, les auteurs décrivent aussi des méthodes *Joint* où une seule particule possède toutes les données à apprendre. En général, ces méthodes semblent moins efficaces que lorsque les données sont réparties sur plusieurs particules (méthodes classiques).

Malheureusement leur implémentation nécessite des outils de développement qui ne sont plus vraiment d'actualité (Theano [11]) ou obsolètes (Rllab [12]).

2.4 Sequential Learning of Agents (SaLiNa)

Dans le cadre de notre projet il est nécesaire d'utiliser des méthodes de décisions séquentielles. En effet, nous utilisons SVPG pour résoudre des problèmes de type "jeu" dont le but est de réagir aux réponses de l'environnement face à nos décisions (ou évolutions naturelles), de facon à maximiser un objectif.

Les bibliothèques de Deep Learning classiques (comme PyTorch [3]) intègrent mal ces méthodes. Elles nécessitent d'implémenter des outils spécifiques à certains cas du RL qui définissent de nouvelles couches d'abstraction, rendant l'utilisation et la prise en main compliquées.

Salina [4] est une bibliothèque récente et prometteuse, basée sur PyTorch, qui a pour objectif de rendre l'implémentation de modèles d'apprentissage séquentiel plus naturelle. Pour ce faire elle se base sur 2 principes :

— tout est un agent, les méthodes s'appliquent donc séquentiellement à tous les agents,

— les agents échangent des informations à travers un *workspace* accessible à tous, facilitant les interactions.

C'est pourquoi, pour moderniser et faciliter l'implémentation de l'algorithme, nous avons choisi de la baser principalement sur SaLiNa, PyTorch, Numpy [13] et Gym [5].

Chapitre 3

Contribution

Nous décrivons maintenant les différentes étapes allant de la compréhension à l'implémentation du projet.

3.1 Besoins préalables

Pour pouvoir se lancer dans le projet plus sereinement, avant de débuter l'implémentation de l'algorithme, nous avons accumuler un certain nombre de connaissances.

Dans un premier temps, afin de mieux comprendre les différents termes techniques, nous avons dû en apprendre plus sur le Machine Learning de manière générale. Les cours de RA ont été particulièrement utiles. Les premiers cours ayant été dispensés par notre encadrant M. Sigaud, il a pu nous donner des explications plus approfondies et nous orienter vers ses cours de M2. Il nous a aussi préparé plusieurs ressources (Google Collabs, slides...) sur SVPG et A2C. Nous avons ensuite étudié l'article de référence [1] afin de comprendre les concepts et les résultats obtenus.

Dans un second, nous avons pris en main les différents outils. Nous avons appris à manipuler PyTorch et Gym à partir des différents exemples de leur documentation. Pour SaLiNa nous avons lu le papier [4], étudier la documentation et suivi une série de vidéos explicatives de l'auteur, Ludovic Denoyer. Ce dernier étant un ancien membre du Lip6, nous avons pu échanger avec lui pour nous éclairer sur certains points.

Enfin, les notions et les outils bien en tête, nous avons commencé le "décryptage" du code de référence à partir du dépôt original et des différents collabs. À partir de là nous avons pu commencer l'implémentation de l'algorithme.

3.2 Implémentation

— implémentation de A2C
— implémentation de SVPG (anneal, svgd kernel, boucle principale)
— wrappers environnements Rllab
— implémentation des outils de visualisation
— astuces utilisées

A expliciter : (afficher du code)

3.3 Analyse des résultats

Chapitre 4

Conclusion

La conclusion de votre rapport doit brièvement résumer ce que vous avez fait, en mettant en lumière les points forts et les points faibles de votre travail. Enfin, il faut décrire les perspectives de poursuite qui peuvent être envisagées.

Bibliographie

[1] Yang Liu, Prajit Ramachandran, Qiang Liu et Jian Peng. « Stein Variational Policy Gradient ». en. In : (avr. 2017). URL : https://arxiv.org/abs/1704.02399v1 (visité le 22/02/2022) (pages 1, 3, 4).

[2] Richard S. Sutton et Andrew G. Barto. *Reinforcement learning : an introduction*. Second edition. Adaptive computation and machine learning series. Cambridge, Massachusetts : The MIT Press, 2018. ISBN : 978-0-262-03924-6 (page 1).

[3] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai et Soumith Chintala. « PyTorch : An Imperative Style, High-Performance Deep Learning Library ». In : *arXiv :1912.01703 [cs, stat]* (déc. 2019). arXiv : 1912.01703. URL : http://arxiv.org/abs/1912.01703 (visité le 03/04/2022) (pages 1, 3).

[4] Ludovic Denoyer, Alfredo de la Fuente, Song Duong, Jean-Baptiste Gaya, Pierre-Alexandre Kamienny et Daniel H. Thompson. « SaLinA : Sequential Learning of Agents ». In : (oct. 2021). arXiv : 2110.07910. URL : http://arxiv.org/abs/2110.07910 (visité le 26/02/2022) (pages 1, 3, 4).

[5] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang et Wojciech Zaremba. « OpenAI Gym ». In : *arXiv :1606.01540 [cs]* (juin 2016). arXiv : 1606.01540. URL : http://arxiv.org/abs/1606.01540 (visité le 03/04/2022) (pages 1, 3).

[6] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver et Koray Kavukcuoglu. « Asynchronous Methods for Deep Reinforcement Learning ». In : *arXiv :1602.01783 [cs]* (juin 2016). arXiv : 1602.01783. URL : http://arxiv.org/abs/1602.01783 (visité le 26/02/2022) (page 1).

[7] Ronald J. Williams. « Simple statistical gradient-following algorithms for connectionist reinforcement learning ». en. In : *Machine Learning* 8.3 (mai 1992), p. 229-256. ISSN : 1573-0565. DOI : 10.1007/BF00992696. URL : https://doi.org/10.1007/BF00992696 (visité le 09/05/2022) (page 1).

[8] David M. Blei, Alp Kucukelbir et Jon D. McAuliffe. « Variational Inference : A Review for Statisticians ». In : *Journal of the American Statistical Association* 112.518 (avr. 2017). arXiv : 1601.00670, p. 859-877. ISSN : 0162-1459, 1537-274X. DOI : 10.1080/01621459.2017.1285773. URL : http://arxiv.org/abs/1601.00670 (visité le 03/04/2022) (page 2).

[9] Qiang Liu et Dilin Wang. « Stein Variational Gradient Descent : A General Purpose Bayesian Inference Algorithm ». In : *arXiv :1608.04471 [cs, stat]* (sept. 2019).

arXiv : 1608.04471. URL : http://arxiv.org/abs/1608.04471 (visité le 03/04/2022) (page 2).

[10] Qiang LIU, Jason D. LEE et Michael I. JORDAN. « A Kernelized Stein Discrepancy for Goodness-of-fit Tests and Model Evaluation ». In : *arXiv :1602.03253 [stat]* (juill. 2016). arXiv : 1602.03253. URL : http://arxiv.org/abs/1602.03253 (visité le 09/05/2022) (page 3).

[11] Rami AL-RFOU et al. « Theano : A Python framework for fast computation of mathematical expressions ». In : *arXiv e-prints* abs/1605.02688 (mai 2016). URL : http://arxiv.org/abs/1605.02688 (page 3).

[12] Yan DUAN, Xi CHEN, Rein HOUTHOOFT, John SCHULMAN et Pieter ABBEEL. « Benchmarking Deep Reinforcement Learning for Continuous Control ». In : *arXiv :1604.06778 [cs]* (mai 2016). arXiv : 1604.06778. URL : http://arxiv.org/abs/1604.06778 (visité le 10/05/2022) (page 3).

[13] Charles R. HARRIS, K. Jarrod MILLMAN, Stéfan J. van der WALT, Ralf GOMMERS, Pauli VIRTANEN, David COURNAPEAU, Eric WIESER, Julian TAYLOR, Sebastian BERG, Nathaniel J. SMITH, Robert KERN, Matti PICUS, Stephan HOYER, Marten H. van KERKWIJK, Matthew BRETT, Allan HALDANE, Jaime Fernández del RÍO, Mark WIEBE, Pearu PETERSON, Pierre GÉRARD-MARCHANT, Kevin SHEPPARD, Tyler REDDY, Warren WECKESSER, Hameer ABBASI, Christoph GOHLKE et Travis E. OLIPHANT. « Array programming with NumPy ». en. In : *Nature* 585.7825 (sept. 2020). Number : 7825 Publisher : Nature Publishing Group, p. 357-362. ISSN : 1476-4687. DOI : 10.1038/s41586-020-2649-2. URL : https://www.nature.com/articles/s41586-020-2649-2 (visité le 10/05/2022) (page 3).

Annexe A

Cahier des charges

Rappel du cahier des charges.

Annexe B

Manuel utilisateur

Un manuel utilisateur permettant la prise en main de votre code.

— code general implementation/ structure