Cahier des charges

Présentation du projet

Contexte. L'apprentissage par renforcement profond a connu des avancées significatives ces dernières années, notamment grâce à des algorithmes comme *SAC* (Soft Actor-Critic) et *TD3* (Twin Delayed Deep Deterministic) qui ont établi l'état de l'art dans le domaine. Ces algorithmes reposent sur une architecture *acteur-critique*, où un réseau de neurones (l'acteur) apprend à sélectionner des actions tandis qu'un autre (le critique) évalue leur qualité.

Dans ce contexte, l'algorithme *AFU* (Actor-Free critic Updates), récemment développé par M. Perrin à l'ISIR, propose une approche innovante qui s'écarte de ce paradigme. AFU se distingue par sa capacité à apprendre sans dépendre explicitement d'un acteur pour la mise à jour du critique, une caractéristique qui pourrait lui conférer des avantages significatifs en termes de stabilité et de généralisation.

Ce projet s'inscrit dans le cadre du parcours AI2D du Master d'Informatique de Sorbonne Université. Sous la direction de M. Sigaud, membre de l'ISIR, nous chercherons à valider expérimentalement les propriétés théoriques d'AFU et à explorer ses avantages potentiels par rapport aux approches traditionnelles.

Objectif de recherche. Notre étude se concentre sur deux hypothèses principales qui, si elles sont validées, pourraient positionner AFU comme une alternative sérieuse aux algorithmes actuels.

(Apprentissage à partir de données aléatoires) La première hypothèse concerne la capacité d'AFU à apprendre à partir de données générées de manière complètement aléatoire. Dans les algorithmes traditionnels d'apprentissage par renforcement, la qualité des données d'apprentissage dépend fortement de la politique d'exploration utilisée. Une limitation majeure des approches actuelles est leur difficulté à apprendre à partir de donnée très éloignées de leur politique courante. AFU pourrait surmonter cette limitation grâce à sa conception qui sépare complètement l'apprentissage du critique de la politique d'exploration.

(Transition hors-ligne/en-ligne) La seconde hypothèse porte sur la stabilité d'AFU lors de la transition entre apprentissage hors-ligne et en-ligne. Les algorithmes actuels souffrent souvent d'une dégradation significative de leurs performances lors de cette transition. Cette dégradation s'explique par le changement brutal dans la distribution des données d'apprentissage. Grâce à sa nature véritablement *off-policy*, AFU pourrait maintenir des performances stables durant cette phase.

Impact attendu. La validation de ces hypothèses aurait un impact positif dans le domaine du Reinforcement Learning :

– une plus grade flexibilité dans la collecte de données d'apprentissage,
– une transition plus fluide entre les deux phrases d'apprentissage hors-ligne et en-ligne.

Ces avancées pourraient être utiles pour des applications robotiques où la collecte de données est coûteuse et la stabilité de l'apprentissage importante.

Cadre technique

Environnement de développement.

- Framework principal: BBRL. BBRL est une bibliothèque conçue à des fins éducatives pour faciliter le codage des algorithmes d'apprentissage par renforcement. L'idée générale est de représenter tout comme des agents interagissant dans un espace de travail, ce qui est similaire à l'approche Blackboard dans les systèmes multi-agents;
- Implémentation en python avec PyTorch. PyTorch est une bibliothèque open-source de machine learning développée par Meta et très utilisée en recherche, notamment dû à sa fonctionalité d'auto-grad. Elle offre une grande flexibilité et une facilité d'utilisation pour le développement d'algorithmes d'apprentissage profond;
- Utilisation de Gymnasium pour les environnements de test. Gymnasium est une suite d'environnements de test pour l'apprentissage par renforcement, développée par OpenAI. Elle offre une grande variété d'environnements pour tester les algorithmes d'apprentissage par renforcement;
- Intégration continue avec Jenkins pour automatiser les tests et les déploiements.;

Implémentation.

- Développement de différents algorithmes de DeepRL avec BBRL et PyTorch. Nous commençons par l'implémentation d'algorithmes simples tels que DQN afin de nous familiariser avec les outils et les environnements de test puis nous passons à des algorithmes plus complexes comme DDPG qui est une version continue de DQN et SAC, qui est basé sur la maximisation de l'entropie. Ces algorithmes servent de base pour comparer les performances d'AFU;
- Adaptation d'environments Gymnasium à notre implémentation. Nous rajoutons des méthodes afin de mettre en place les tirages d'actions uniformes dans les environnements;
- Mise en place d'outils de mesure de performance et de visualisation des résultats;

Protocole Expérimental

Étude de l'apprentissage aléatoire.

- Modification des environnements pour permettre un échantillonnage uniforme;
- Comparaison des performances d'AFU avec celles de SAC et DDPG dans des scénarios d'apprentissage à partir de données aléatoires;

Analyse de la transition hors-ligne/en-ligne.

- Phase d'apprentissage hors-ligne avec données pré-collectées;
- Transition progressive vers l'apprentissage en-ligne;
- Suivi continu des performances pendant la transition, comparaison avec Cal-QL et d'autres approches pour améliorer la transition hors-ligne/en-ligne;

Environnemnets de test.

- Cartpole (version continue): Un poteau attaché à un chariot allant de gauche à droite avec pour but de stabiliser le poteau en haut à la verticale. Nous utilisons cet environnement trivial pour tester les premières implémentations d'algorithmes tel que DDPG et SAC;
- Pendulum: Un pendule dont le but est de se placer en haut à la verticale. Nous utilisons cet environment pour tester les performances des algorithmes implémentés;
- Lunar Lander (version continue): Un alunisseur dont le but est de se placer correctement dans la zone d'alunissage. Nous utilisons cet environment pour tester l'apprentissage « purement off-

policy » car l'espace d'action de cet environnement est grand. Egalement, il servira à tester la résilience des algorithmes face aux maxima locaux;
- Swimmer: Plusieurs segments reliés dont le but est de se déplacer le plus rapidement possible. Cet environnement permet de mettre en difficulté DDPG et SAC afin de vérifier qu'AFU est meilleur;
- Ant: Environnement simulant une fourmi dont le but est d'atteindre l'arrivé d'un labyrinthe en forme de U. Cet environnement permet d'évaluer le passage de off-policy à on-policy;
- Autres environnemnets MuJoCo selon les besoins;

Méthodologie d'évaluation

Métrique de performances.
- Efficacité d'apprentissage;
- Stabilité pendant la transition;
- Qualité de la politique finale;

Analyse comparative.
- Comparaison avec SAC, TD3 et IQL;
- Analyse statistique des résultats;
- Visualisation des courbes d'apprentissage;

Organisation du projet

Planning.

Phase 1 :
- Mise en place de l'environnement, création d'un repository sur Github et d'une pipeline d'intégration continue Jenkins;
- Implémentation de différents algorithmes tels que DQN, DDPG et SAC afin de prendre la main sur PyTorch et BBRL ainsi que d'avoir une base comparative pour AFU;
- Implémentation d'AFU from scratch avec BBRL ainsi qu'une adaptation de AFU crée par M. Perrin afin de commencer les tests rapidement et de comparer les deux versions;
- Développement des outils de tests, implémentation des premiers environments Cartpole et Pendulum, mise en place de commandes permettant de lancer des entrainements et de créer des graphes rapidement;

Phase 2 :
- Suite de l'implémentation d'environments tels que Lunar Lander, Swimmer et Ant afin d'étendre les possibilités d'expériences;
- Réalisation des expériences sur les environments nouvellement implémentés
- Collecte des données de test pour DDPG, SAC et AFU;
- Analyses préliminaires des résultats obtenus;
- Réglage des hyperparamètres d'AFU;
- Continuation de l'implémentation d'AFU BBRL;

Phase 3 :
- Analyse approfondie des expériences. Examen des courbes d'apprentissage, comparaison des performances entre AFU et les autres algorithmes et identification des points forts et des faiblesses d'AFU;

– Analyses supplémentaires pour explorer des aspects spécifiques des performances d'AFU, tels que l'impact des hyperparamètres, la robustesse face à des variations dans les environnements de test, et la généralisation à de nouveaux environnements;
– Rédaction du rapport final incluant une description complète de la méthodologie, des résultats expérimentaux, des discussions sur les implications des résultats, et des recommandations pour les travaux futurs;
– Préparation de la soutenance, présentation résumant les objectifs, la méthodologie, les résultats clés, et les conclusions du projet.

Livrables.
– Code source documenté : Le code source complet et bien documenté des implémentations d'AFU et des autres algorithmes, ainsi que des scripts utilisés pour les expériences;
– Résultats expérimentaux : Les données brutes et les analyses des résultats expérimentaux, y compris les courbes d'apprentissage et les statistiques comparatives;
– Rapport final : Un document détaillé présentant l'ensemble du projet, de la conception à l'analyse des résultats;
– Présentation pour la soutenance : Une présentation complète du projet à montrer lors de la soutenance;

Critères de réussite

Le projet sera considéré comme réussi s'il :
1. démontre clairement les capacités ou non d'AFU à apprendre à partir de données aléatoires,
2. quantifie la stabilité d'AFU pendant la transition hors-ligne/en-ligne,
3. fournit des résultats reproductibles,
4. livre une implémentation robuste d'AFU dans BBRL.