

Étude de "Deep Reinforcement Learning From Human Preferences", l'apprentissage par renforcement profond à l'aide de préférences humaines

MÉLANIE KARLSEN, FANNY STREIFF

Février 2020

L'apprentissage par renforcement (RL) est un domaine du machine learning qui se détache des méthodes classiques supervisées et non supervisées. Si le Machine learning (ML) classique apprend sur des données obtenues au préalable, le reinforcement learning lui, apprend à coup d'essai et d'erreur. L'essor actuellement visible dans le domaine du deep learning a su s'appliquer au RL dans de nombreux domaines; entre autres - La conduite automatique, la robotique [2]. Un des problèmes modernes majeurs du RL est le besoin d'avoir une fonction de récompense définie pour pouvoir appliquer la plupart des méthodes classiques [9]. Deep Reinforcement Learning from Human Preferences [3] propose un moyen d'outrepasser ce problème en créant une fonction de récompense à l'aide de supervision humaine. L'idée présentée par Christiano et al. est de pouvoir apprendre de nouveaux comportements à un agent sans avoir accès à une fonction de récompense explicite, permettant ainsi d'appliquer le Reinforcement Learning à des tâches plus compliquées.

1. INTRODUCTION

L'Apprentissage par Renforcement est un domaine du Machine Learning qui sort du modèle classique de l'apprentissage classique supervisé et non supervisé. Le RL fonctionne de la manière suivante : un Agent A peut exécuter différentes actions $\mathcal{A} = \{a_0, \dots, a_n\}$ en suivant une police π sur un environnement E muni d'états $S = \{s_1, \dots, s_t\}$, pour chacune de ses actions, il reçoit un signal, une récompense R , qu'il essaiera de maximiser afin de trouver les meilleures actions à suivre avec un environnement et un état donné. A un état S_0 est prise une action A_0 , qui renvoie une récompense R_0 et un état S_1 , et ainsi de suite. On pourrait faire une analogie avec des expériences classiques de neuroscience : un rat ayant le choix d'appuyer sur deux boutons, l'un lui envoyant un choc électrique, l'autre relâchant de l'endorphine, le rat apprendra à maximiser sa récompense en agissant sur l'environnement via l'action d'appuyer sur le second bouton. Le RL, en explorant les différentes combinaisons d'états et d'actions, pourra trouver les meilleurs comportements.

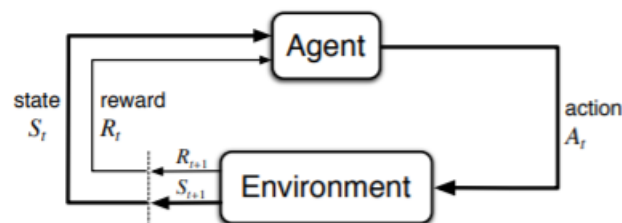


Fig. 1. L'Agent choisit une action A_t qui agit sur l'environnement, qui renvoie un nouvel état S_t et R_t dépendant de l'action prise. Graphique de [9]

Le RL a été utilisé avec succès dans diverses applications; Du célèbre AlphaGO [8] qui a appris le jeu de GO en jouant contre lui même jusqu'à être capable de battre le champion du monde - [6], un agent capable de battre les champions du monde de Dota2, un jeu vidéo en temps réel - où, pour une application plus utile, [7], des méthodes de conduite autonome. On peut aussi citer le succès du RL dans la médecine et la finance.

Les méthodes classiques de maximisation de la récompense (MCMC, TD-Learning, Policy Improvement) ont besoin d'une fonction de récompense explicite. Cette fonction n'est pas difficile à trouver dans un bon nombre d'applications. Le jeu des échecs par exemple a des règles bien précises et modélisables, c'est le cas de Gridworld aussi : trouver la bonne "police" dans un jeu vidéo revient simplement à maximiser le score. Le RL a donc naturellement prospéré dans ces milieux. Le soucis vient donc bien lorsque nous voulons appliquer du RL sur des problèmes nouveaux, difficilement modélisables. Un robot qui voudrait apprendre à faire l'opération de l'appendicite, ou plus simplement promener son chien, n'aura pas de fonction de récompense explicite à maximiser. D'emblée, l'idée principale était de construire une fonction qui maximiserait un comportement ressemblant.

Une première manière d'adresser ce problème partait du postulat suivant : si nous pouvons observer un comportement et en extraire son essence, nous pourrions en inférer une fonction de récompense exacte. Cette idée fut développée au début du millénaire par Andrew Ng et Stuart Russel dans le papier "Algorithms for Inverse Reinforcement Learning" [5]. Avec une mesure d'un agent dans différentes circonstances sur un temps assez long, ainsi qu'un modèle de l'environnement, l'explicitation de la fonction de récompense est faisable. Une application qu'Andrew Ng a faite de cette idée est d'utiliser le RL pour apprendre à un hélicoptère à exécuter différentes figures. Un problème vient cependant contrecarrer le succès de cette méthode : il faut un investissement conséquent en temps d'experts en la matière, ainsi qu'une bonne modélisation de l'environnement. De plus, aucun comportement impossible à réaliser pour l'être humain ne peut être appris de cette manière.

L'investissement important en temps d'experts est un problème pour le RL qui a justement pour but de s'en passer. Christiano et al [3] présentent dans leur papier une manière d'apprendre de nouveaux comportements à des agents, non modélisable par une fonction de récompense, avec une supervision humaine minimale. L'idée principale étant que, bien qu'un humain ne peut pas nécessairement réaliser une action, il peut cependant reconnaître si elle est bien faite ou non. La méthode décrite dans "Deep Reinforcement Learning from human Preferences" propose d'apprendre à un agent de nouveaux comportements avec une fonction de récompense, celle-ci étant apprise à l'aide de retours d'humains sous forme de comparaison de deux courtes vidéos décrivant différents comportements. Cette méthode, bien que déjà étudiée avant, n'a jamais pu être mise à l'échelle auparavant et a toujours eu besoin d'un très gros apport en temps du superviseur.

L'algorithme présenté est simple, il suffit de modéliser une fonction de récompense en tenant compte de la préférence du superviseur humain, tout en entraînant une police afin de maximiser la fonction de récompense courante.

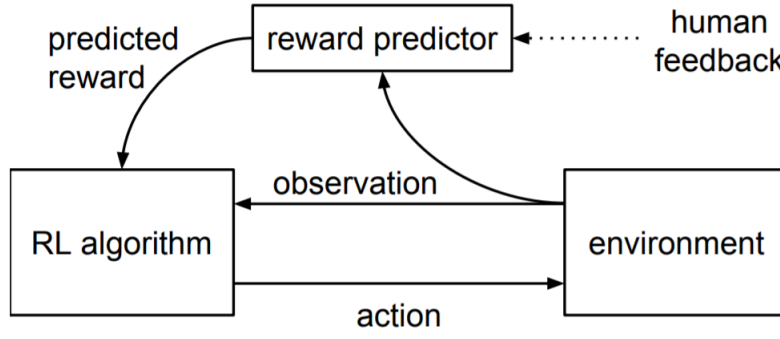


Fig. 2. La prédiction de récompense est entraînée de manière asynchrone en comparaison des préférences sur les segments vidéos. L’agent maximise ensuite la récompense prédite.

Dans ce papier, nous allons donc présenter l’idée principale de Christiano et al. qui a permis de mettre à l’échelle le RL à l’aide de supervision humaine, en présentant les méthodes. Nous allons ensuite expliquer les résultats des applications obtenues dans ce papier, puis nous finirons par expliquer comment nous avons appris un nouveau comportement à un robot sous MuJoCo.

2. LA MÉTHODE DE CHRISTIANO ET AL.

L’idée principale est d’optimiser une fonction de récompense qui s’apprend via des interactions entre le robot et les humains, celles-ci se faisant sous forme de retours de préférence humaine (notion qui sera définie bientôt). Cette technique permet notamment aux robots d’être formés par des gens n’ayant peu ou pas d’expérience avec le machine learning, les retours nécessaires à l’apprentissage étant très faciles à rendre. Définissons certaines notations et certains termes de la même manière que Christiano et al l’ont fait dans leur article, afin de rester cohérent.

Notations pour une observation et une action Soit $o_t \in \mathcal{O}$ l’observation d’un humain au moment t et soit $a_t \in \mathcal{A}$ l’action envoyé par le robot au moment t .

Définition d’un segment de trajectoire Un segment de trajectoire est une séquence d’observations et d’actions dénotée par σ , telle que $\sigma = ((o_0, a_0), \dots, (o_{k-1}, a_{k-1})) \in (\mathcal{O} \times \mathcal{A})^k$

Notation pour l’ordre de préférence humaine On écrit $\sigma^1 \succ \sigma^2$ pour indiquer que l’humain préfère le segment de trajectoire σ^1 au segment de trajectoire σ^2

Définition On dit que les préférences \succ sont générées par une fonction de récompense $r : \mathcal{O} \times \mathcal{A} \rightarrow \mathbb{R}$ si

$$\begin{aligned}
 r(o_0^1, a_0^1) + \dots + r(o_{k-1}^1, a_{k-1}^1) &> r(o_0^2, a_0^2) + \dots + r(o_{k-1}^2, a_{k-1}^2) \\
 \Rightarrow \\
 ((o_0^1, a_0^1), \dots, (o_{k-1}^1, a_{k-1}^1)) &\succ ((o_0^2, a_0^2), \dots, (o_{k-1}^2, a_{k-1}^2))
 \end{aligned}$$

La méthode est donc la suivante : à chaque moment t , une police $\pi : \mathcal{O} \rightarrow \mathcal{A}$ et une estimation de la fonction de récompense $\hat{r} : \mathcal{O} \times \mathcal{A} \rightarrow \mathbb{R}$, paramétrées par des réseaux de neurones profonds, sont mises à jour via le procédé suivant :

1. La police π interagit avec l’environnement pour produire un ensemble de trajectoires $\{\tau^1, \dots, \tau^i\}$. Les paramètres de π sont mis à jour à l’aide d’un algorithme de RL traditionnel, afin de maximiser la somme des récompenses prédites $r_t = \hat{r}(o_t, a_t)$.
2. Des paires de segments (σ^1, σ^2) sont sélectionnées des trajectoires $\{\tau^1, \dots, \tau^i\}$ produites à l’étape 1, et sont envoyées à un humain pour que comparaison soit faite.

3. Les paramètres de \hat{r} sont optimisés par un algorithme d'apprentissage supervisé afin de correspondre un maximum aux comparaisons faites par l'humain.

À noter que ces étapes se déroulent de manière asynchrone mais continue. Dans leur article, Christiano et al. dédient un paragraphe explicatif entier à chaque étape du processus. Nous allons reprendre ce même format puisque cette explication est nécessaire à la compréhension globale de leur méthode.

A. Étape 1 - L'optimisation de la police

Il s'agit là d'une étape très typique de RL traditionnel, puisqu'à ce moment-là de l'algorithme, un estimateur de la fonction de récompense est connu. Cependant, la fonction de récompense \hat{r} peut être non-stationnaire, ce qui nous oblige alors à préférer des méthodes robustes aux changements de cette fonction. C'est pourquoi Christiano et al. se sont concentrés sur les méthodes de gradient de police, dont la preuve du bon fonctionnement a été faite dans un article de Jonathan Ho and Stefano Ermon. Plus précisément, ils ont utilisé la méthode A2C (advantage actor-critic) et TRPO (trust region policy optimization), respectivement pour le jeu Atari et pour exécuter d'autres types de tâche. Dans chacun des cas, le paramétrage a été fait de manière traditionnelle pour des problèmes de RL, à l'exception d'un hyperparamètre dans le cas du TRPO, l' "entropy bonus", qu'ils ont eux-mêmes ajusté.

A.1. L'A2C : Advantage actor critic method

Une des deux méthodes utilisées pour l'optimisation de la fonction de récompense est l'A2C, présentée par Mnih et al. [4]. Cette méthode est appliquée sur les jeux Atari et présente une descente de gradient asynchrone applicable au deep reinforcement learning. La méthode usuelle d'apprentissage pour le RL stocke les données de l'agent dans la mémoire, ce qui est loin d'être optimal car le coût en mémoire et en calcul est important, nécessitant donc l'apprentissage à l'aide de méthode "off policy". La méthode présentée par [4] outrepassse ce problème en faisant passer plusieurs agents de manière simultanée sur des environnements en parallèle. On représente la fonction valeur/action par une approximation dépendant d'un paramètre θ qui peut être obtenu avec du Q-Learning. On note aussi $b_t(s_t) = V^\pi(s_t)$ en tant que baseline pour réduire la variance du gradient de la police. Utiliser $R_t - b_t$ est une estimation de la fonction avantage de l'action a_t en l'état s_t $A(a_t, s_t) = Q(a_t, s_t) - V(s_t)$ où R_t est une estimation de la fonction action/valeur. C'est une architecture de type acteur/critique, avec la police π en acteur et la baseline b_t la critique. L'algorithme utilisé par [3] est "l'A3C". Cela maintient une police $\pi(a_t|s_t; \theta)$ et une estimation de la fonction valeur $V(s_t; \theta_v)$. Une méthode à n-pas est utilisée et renvoie une modification de la fonction police et valeur.

Algorithm S3 Asynchronous advantage actor-critic - pseudocode for each actor-learner thread.

// Assume global shared parameter vectors θ and θ_v and global shared counter $T = 0$

// Assume thread-specific parameter vectors θ' and θ'_v

Initialize thread step counter $t \leftarrow 1$

repeat

Reset gradients: $d\theta \leftarrow 0$ and $d\theta_v \leftarrow 0$.

Synchronize thread-specific parameters $\theta' = \theta$ and $\theta'_v = \theta_v$

$t_{start} = t$

Get state s_t

repeat

Perform a_t according to policy $\pi(a_t|s_t; \theta')$

Receive reward r_t and new state s_{t+1}

$t \leftarrow t + 1$

$T \leftarrow T + 1$

until terminal s_t **or** $t - t_{start} == t_{max}$

$R = \begin{cases} 0 & \text{for terminal } s_t \\ V(s_t, \theta'_v) & \text{for non-terminal } s_t // \text{ Bootstrap from last state} \end{cases}$

for $i \in \{t - 1, \dots, t_{start}\}$ **do**

$R \leftarrow r_i + \gamma R$

Accumulate gradients wrt θ' : $d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi(a_i|s_i; \theta')(R - V(s_i; \theta'_v))$

Accumulate gradients wrt θ'_v : $d\theta_v \leftarrow d\theta_v + \partial (R - V(s_i; \theta'_v))^2 / \partial \theta'_v$

end for

Perform asynchronous update of θ using $d\theta$ and of θ_v using $d\theta_v$.

until $T > T_{max}$

B. Étape 2 - L'explicitation des préférences

Le choix des préférences se fait de la manière suivante :

1. Deux visualisations de deux segments de trajectoire sont présentées à un humain sous forme d'un clip vidéo de une à deux secondes chacune.
2. Dans le cas où un clip est visiblement mieux que l'autre, l'humain indique à l'agent quel clip il préfère. Dans l'autre cas, soit il indique à l'agent que les deux clips semblent tous deux exécuter la tâche tout aussi bien, ou bien qu'il n'est pas en mesure de comparer les deux.
3. Le jugement de l'humain est enregistré dans une base de données \mathcal{D} de triplets $(\sigma^1, \sigma^2, \mu)$ où σ^1 et σ^2 sont les deux segments qui ont été comparés et μ est une distribution sur $\{1, 2\}$ indiquant quel segment l'humain a préféré. Si l'humain préfère un segment à l'autre, alors μ met tout son poids sur celui-ci. S'il trouve les deux tout aussi bien l'un que l'autre, alors μ est uniforme. Dans le cas échéant, la comparaison n'est tout simplement pas incluse dans la base de données.

C. Étape 3 - L'apprentissage de la fonction de récompense

On peut considérer que le but ultime est de minimiser l'erreur de prédiction. Dans leur papier, Christiano et al. décident donc de minimiser l'entropie croisée. De fait, celle-ci s'exprime de la manière suivante :

$$\text{loss}(r) = - \sum_{(\sigma^1, \sigma^2, \mu) \in \mathcal{D}} \mu(1) \log \mathbb{P}[\sigma^1 \succ \sigma^2] + \mu(2) \log \mathbb{P}[\sigma^2 \succ \sigma^1]$$

où :

$$\mathbb{P}[\sigma^1 \succ \sigma^2] = \frac{\exp \sum r(\sigma_t^1, a_t^1)}{\exp \sum \hat{r}(\sigma_t^1, a_t^1) + \sum r(\sigma_t^2, a_t^2)}$$

Puisque nous travaillons avec \hat{r} , un estimateur de r , nous reformulons le problème de la manière suivante :

$$\text{minimiser } \text{loss}(\hat{r}) = - \sum_{(\sigma^1, \sigma^2, \mu) \in \mathcal{D}} \mu(1) \log \hat{\mathbb{P}}[\sigma^1 \succ \sigma^2] + \mu(2) \log \hat{\mathbb{P}}[\sigma^2 \succ \sigma^1]$$

où :

$$\hat{\mathbb{P}}[\sigma^1 \succ \sigma^2] = \frac{\exp \sum \hat{r}(\sigma_t^1, a_t^1)}{\exp \sum \hat{r}(\sigma_t^1, a_t^1) + \sum \hat{r}(\sigma_t^2, a_t^2)}$$

Il est important de noter qu'il s'agit d'un modèle de Bradley-Terry d'estimation de fonction selon une préférence, modifiée par l'idée du choix de Luce-Shepard. Ces modèles ont pour but de prédire le résultat d'une comparaison de jeu i, j d'une population où $P(i > j) = \frac{p_i}{p_i + p_j}$. Ce type de modèle est à l'origine du score Elo.

Concrètement, l'apprentissage de la fonction r se fait de la façon suivante :

- Plusieurs estimateurs \hat{r} sont entraînés, chacun d'eux étant entraînés sur $|\mathcal{D}|$ échantillons de \mathcal{D} avec remplacement. L'estimateur \hat{r} est défini en normalisant et centrant tous ces estimateurs.
- Une fraction $1/e$ des données est mise de côté afin d'être utilisée pour la validation de chaque estimateur. La régularisation l_2 est utilisée et le paramètre de régularisation λ est ajusté afin de garder la loss de validation entre 1.1 à 1.5 fois la loss d'entraînement.
- L'utilisation de la fonction softmax génère un problème : en effet, elle ne tient pas compte de la probabilité d'erreur humaine. Une façon de la prendre en compte est d'ajouter un terme stochastique, partant du principe que l'homme répond 10% du temps de façon aléatoire (et uniforme).

D. La sélection des queries

L'apprentissage est fait avec de l'ensemble learning. Plusieurs prédicteurs sont entraînés en parallèle. L'idée est de prendre en priorité les queries qui sont débattues par les différents prédicteurs. On échantillonne donc k paires de trajectoires, et on prend en priorité les trajectoires qui ont la plus grande variance entre les prédicteurs.

3. NOTRE EXPÉRIMENTATION

Pour la démarche scientifique, nous avons voulu apprendre un nouveau comportement à un robot sous MuJoCo en utilisant l'implémentation disponible [1]. L'idée était d'entraîner le ant à marcher. Il s'agit d'un problème assez simple qui a d'ailleurs une fonction de récompense bien définie. Chaque segment devait être choisi de la manière suivante :

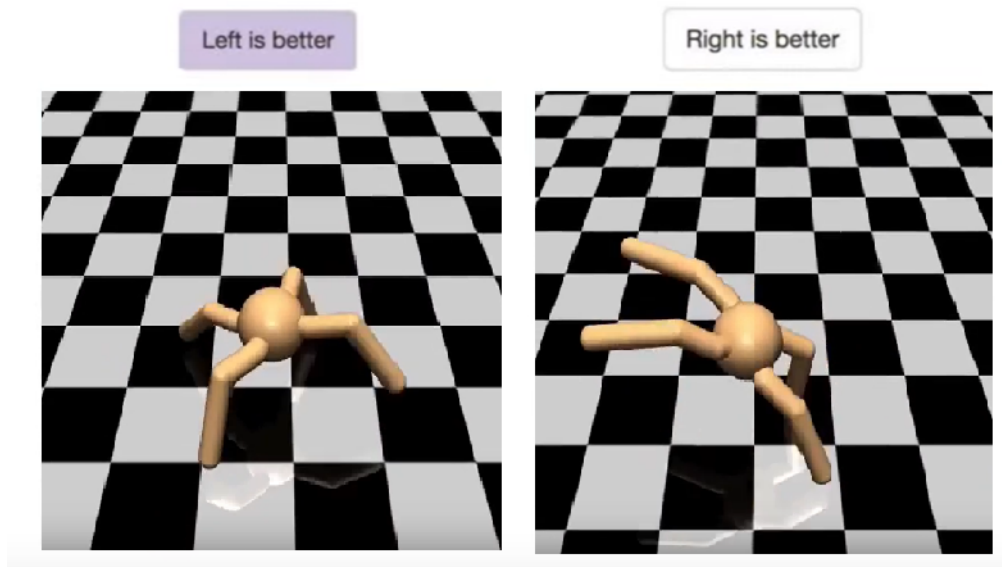


Fig. 3. Le programme nous propose deux courts segments vidéos d'un comportement du robot. Ici, nous souhaitons que le robot marche, le choix de gauche est donc préférable.

L'entraînement procède de la manière décrite. Les deux segments vidéos sont proposés, et selon les comportements pris par les robots, le superviseur humain fait son choix. Au départ, les segments vidéos vont proposer des comportements complètement erratiques. Petit à petit, à l'aide du superviseur humain, la fonction récompense se forme aux comportements préférés. En lien seront trois gifs représentant l'entraînement du robot au début, après 30 minutes, puis 1h15 de supervision. On notera qu'en seulement 1h15 de supervision humaine et sans fonction de récompense définie, le robot apprend à marcher correctement.

4. APPLICATIONS

Maintenant que nous avons testé et montré le procédé derrière l'entraînement d'un nouveau comportement selon les méthodes de [3], nous allons présenter les résultats importants quantitativement obtenus par le papier originel. Dans le papier, plusieurs tâches sont réalisées sur des comportements dont la fonction récompense est connue ou non. Il est important de faire cette distinction, car dans le cas où la fonction récompense est connue, nous pouvons ainsi obtenir une comparaison avec les méthodes traditionnelles d'optimisation du RL, que nous essayons **d'égaliser**. En effet, obtenir des résultats tout aussi bons lorsque la fonction récompense est inconnue à lorsqu'elle est connue, est plus que satisfaisant. Chacune des expériences a eu besoin de 30min à 5 heures de supervision humaine.

A. Comparaison avec des comportements à récompense explicite connue.

Nous allons d'abord présenter les résultats obtenus sur les comportements explicites connus, que ce soit pour la simulation sous MuJoCo ou pour les jeux Atari. L'expérience est faite suivant ce schéma : muni des résultats obtenus avec les méthodes classiques de RL, est créé un oracle. Des queries seront envoyées à l'oracle, qui simulera donc le choix du superviseur humain. Ayant de plus accès à la fonction reward exacte, l'oracle choisira systématiquement le segment qui optimisera cette dernière. Il comparera 300, 700, à 1400 queries de la même manière qu'un superviseur humain. L'oracle ayant cependant accès à la fonction de récompense exacte, il choisira systématiquement le meilleur segment. Cet oracle ainsi que la baseline de RL sera aussi comparée au résultat obtenu avec 750 queries envoyées à un humain, qui devra choisir selon les deux segments avec une description de deux phrases de la tâche à accomplir.

Dans le cas où la fonction de récompense est connue, nous allons regarder trois comportements différents obtenus. Deux sous MuJoCo, et un avec Atari. Le premier est le cas de Walker. La tâche est simple, il suffit à un robot avec trois degrés de libertés de marcher vers la droite, le second sera ant, qui est une expérience similaire à celle que nous avons entraînée avant.

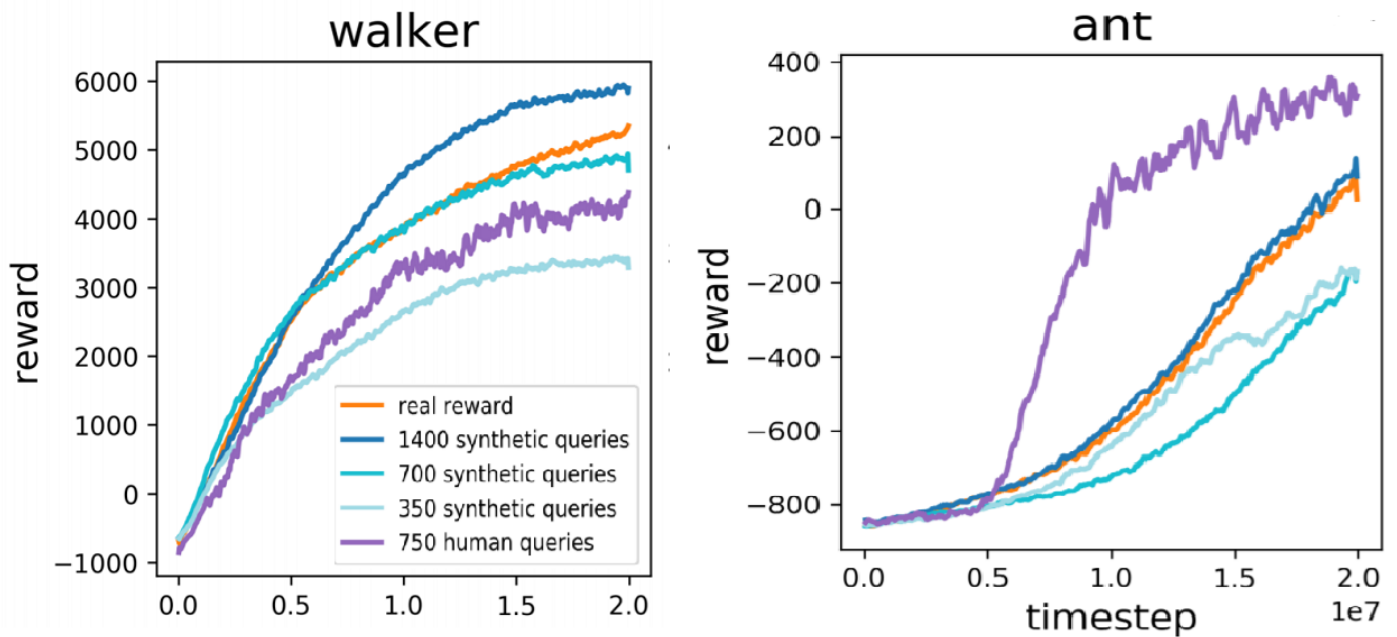


Fig. 4. Comparaison de la récompense obtenue avec la vraie reward, ainsi que des query synthétiques et de la supervision humaine pour deux tâches sous MuJoCo.

Les résultats obtenus pour la tâche "walker" sont généraux. Il est intéressant de remarquer que souvent, les 1400 queries synthétiques obtiennent un meilleur résultat que ce qui est obtenu avec les méthodes traditionnelles de l'apprentissage par renforcement. Cela vient certainement du fait que la fonction récompense est mieux formée. Dans tous les cas sauf pour la tâche "Reacher", les 1400 queries obtenaient un résultat meilleur que le RL traditionnel, et ce, *alors que la fonction récompense est inconnue*. Encore plus intéressant, même avec la vraie reward connue, l'apprentissage avec supervision humaine est plus efficace que le RL traditionnel et la supervision synthétique dans certaines tâches !

Un dernier cas est intéressant, et ce dans une tâche spécifique sur Atari. Sur Atari, les humains devaient réaliser 5500 comparaisons, l'oracle avait 3.3, 5.6, 10k comparaisons. Dans une tâche spécifique (Enduro), les résultats observés furent les suivants :

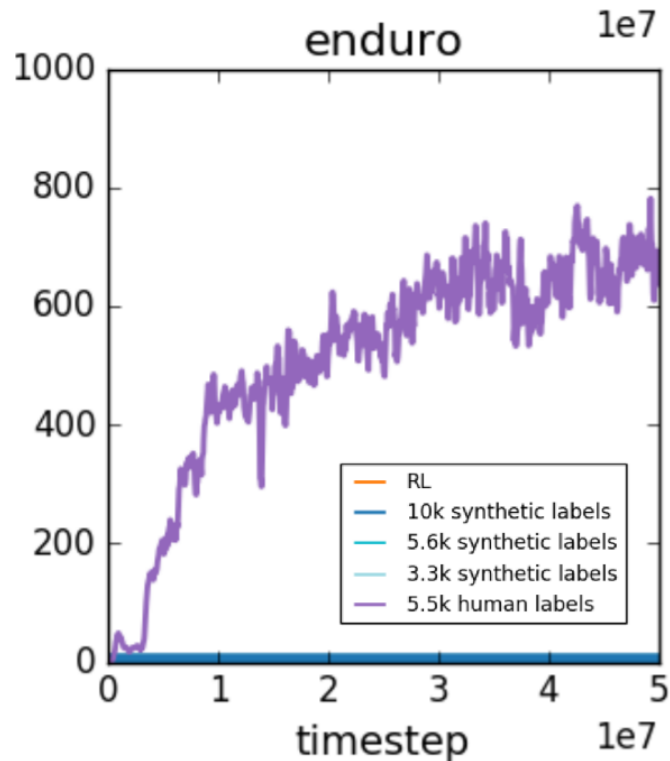


Fig. 5. Résultats obtenu sur la tâche Atari Enduro

Ici, on peut voir que *seulement* la supervision humaine a pu mener à bien la tâche. En effet, la supervision humaine récompensait un comportement qui n'était pas dans la fonction de récompense exacte, ce qui a permis l'obtention de résultats bien supérieurs aux autres méthodes, même celles du RL classique. C'est précisément dans ces cas-là que la supervision humaine trouverait son point fort.

A.1. Les nouveaux comportements

La comparaison avec le RL traditionnel est très utile. Il a été possible d'observer que la méthode développée ici égale souvent et surpasse même parfois le RL traditionnel sur une fonction connue. L'intérêt principal de cette méthode reste cependant à être montré quantitativement. Notamment dans le cas de la création de mouvements jamais fait auparavant en RL, à l'aide de supervision humaine. Ici aussi, Christiano et al. [3] montre qu'ils ont pu apprendre de nouveaux types de comportements aux robots. Le robot hopper a pu, par exemple, apprendre à enchaîner les salto arrières, en moins d'une heure, le robot guépard à marcher sur une jambe, et sur le jeu Enduro, à rester entre les voitures.

A.2. Etudes des comportements par ablation

Une dernière partie compte des modifications des comportements.

- Les queries sont prises de manière uniforme au lieu de prioriser les queries où les différents prédicteurs ne sont pas d'accord.
- Au lieu de procéder par ensemble training, un seul prédicteur est entraîné. Les queries sont de plus prises aléatoirement de manière uniforme, étant donnée que deux prédicteurs ne peuvent pas donner des résultats contradictoires.
- L'entraînement est fait seulement sur les queries obtenues en début d'entraînement.
- L'entraînement est réalisé sans régularisation.

- Au lieu de créer \hat{r} avec les comparaisons, un oracle qui donne la récompense totale sur un segment est utilisé pour calculer \hat{r} .

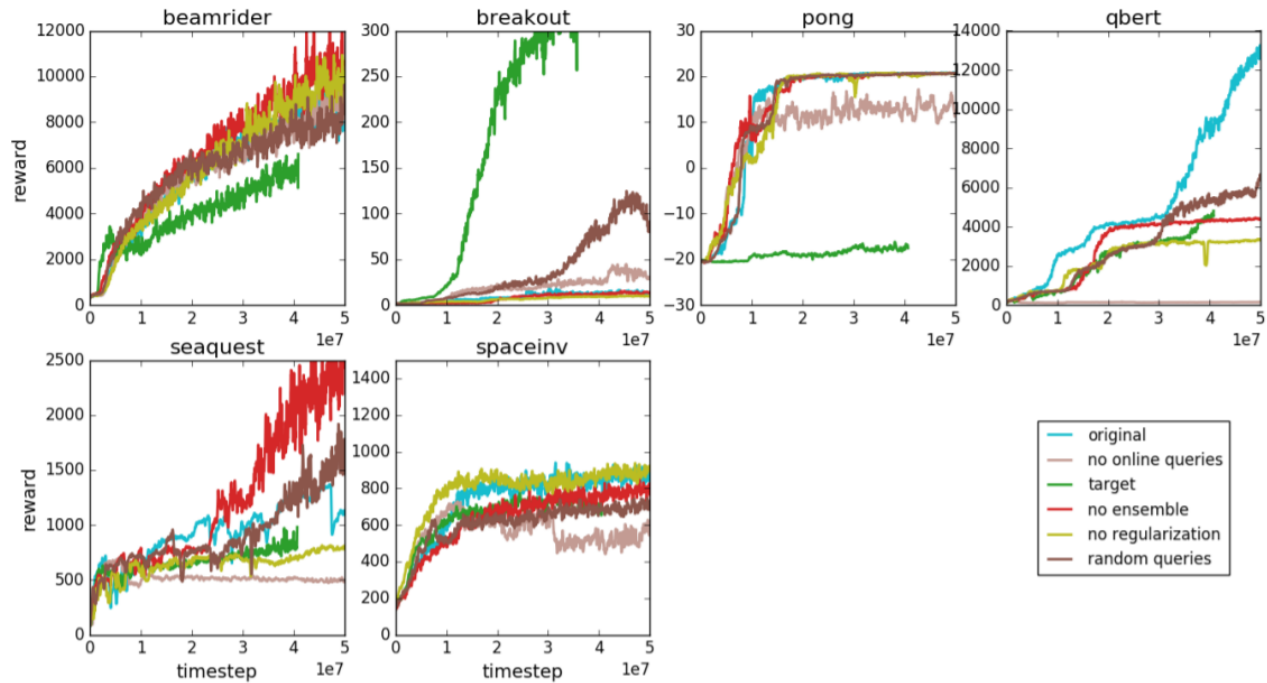


Fig. 6. Les performances sur les entraînements avec ablation.

On peut remarquer que, sans surprise, avec un entraînement sur les queries de base, le résultat est toujours mauvais. Le prédicteur ne capture donc qu'une partie de la vraie fonction de récompense. On remarque aussi que prendre des queries de manière uniforme aide parfois à l'apprentissage, tout comme le fait d'éviter de prendre plusieurs classifieurs.

5. CONCLUSION

Nous avons pu voir que [3] a créé une méthode efficace pour généraliser l'apprentissage par renforcement à des comportements complexes. Cette méthode a non seulement l'avantage d'être applicable dans des milieux qui n'ont auparavant pas été pu être exploités par le RL, mais a en plus su le faire d'une manière efficace. L'absence de nécessité des superviseurs humains experts pendant une longue période de temps, comme pour les anciennes méthodes, permettra au RL de trouver un nouvel essor. Une limite reste que certaines tâches peuvent être reconnues par un être humain mais ne pourront être correctement évaluées que par un expert. Même dans ce cas là, la rapidité de l'apprentissage de la fonction de récompense permettrait d'appliquer la méthode.

REFERENCES

1. Github, l.. Accessed: 2010-09-30.

Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. A brief survey of deep reinforcement learning. *CoRR*, abs/1708.05866, 2017. URL <http://arxiv.org/abs/1708.05866>.

Paul Christiano, Jan Leike, Tom B. Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. 2017.

Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *CoRR*, abs/1602.01783, 2016. URL <http://arxiv.org/abs/1602.01783>.

Andrew Ng and Stuart Russell. Algorithms for inverse reinforcement learning. *ICML '00 Proceedings of the Seventeenth International Conference on Machine Learning*, 05 2000.

OpenAI, :, Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique Pondé de Oliveira Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. Dota 2 with large scale deep reinforcement learning, 2019.

Shai Shalev-Shwartz, Shaked Shammah, and Amnon Shashua. Safe, multi-agent, reinforcement learning for autonomous driving. *CoRR*, abs/1610.03295, 2016. URL <http://arxiv.org/abs/1610.03295>.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dhharshan Kumaran, Thore Graepel, Timothy P. Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *CoRR*, abs/1712.01815, 2017. URL <http://arxiv.org/abs/1712.01815>.

Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998. ISBN 0262193981.