MASTER MSV

Étude d'article : Deep Reinforcement Learning from Human Preferences

Yasmine DAOUIA, Taliesin ROMANET & Florian VERDIER

lien vers le code :
`https://github.com/pelouse/Human-Preferences-on-maze`

Contents

1 Introduction

L'article étudié, *"Deep Reinforcement Learning from Human Preferences"*, fut publié en 2017 et est le fruit du travail de 6 auteurs, principalement issus de DeepMind et OpenAI. Il part du constat que, jusqu'alors, la majorité des avancées en apprentissage par renforcement étaient effectués avec des fonctions de récompense bien définies et s'attache à explorer une méthode pour dépasser cette limite et effectuer de l'apprentissage par renforcement même en l'absence de fonction de récompense, pourtant nécessaire classiquement. Le point clé de l'article est ainsi d'utiliser des retours d'un utilisateur évaluant les performances de l'algorithme à chaque étape pour obtenir une fonction de récompense estimée ayant les bonnes caractéristiques pour obtenir le comportement attendu de l'agent entraîné.

1.1 Vocabulaire

Dans toute la suite de ce rapport, on considère le vocabulaire introduit dans le cours de Reinforcement Learning. On va ainsi considérer un *agent* en contact avec un *environnement* avec lequel il peut interagir. Pour cela, il effectue des *actions* suivant une *politique*, qui est une fonction, déterministe ou non, de l'état actuel de l'environnement. L'environnement répond alors à cette action par une *récompense* et un nouvel état de l'environnement. Ceci est résumé dans le schéma 1.

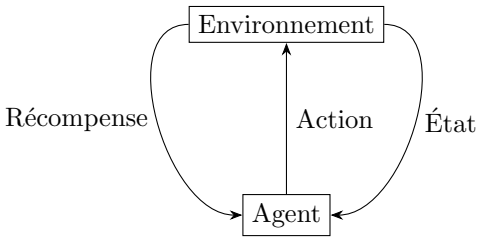


Figure 1: Schéma de fonctionnement usuel de l'apprentissage par renforcement

1.2 Cadre de l'article

Afin d'effectuer de l'apprentissage par renforcement, il est indispensable de disposer d'une fonction de récompense. C'est en effet ce qui permet d'indiquer à l'agent la performance des choix qu'il a effectués afin qu'il puisse faire évoluer en conséquence sa politique et ainsi la faire converger vers une politique optimale.

Dans certains cas, il est relativement aisé de choisir une telle fonction. On peut ainsi penser à un agent apprenant à jouer aux échecs et pour lequel une fonction de récompense simple serait la différence de valeur des pièces présentes sur l'échiquier entre l'agent et son adversaire (en comptant par exemple 1 point

pour un pion, 9 points pour une dame, etc...[1])

Au contraire, une fonction de récompense appropriée peut être plus complexe à imaginer dans d'autres situations. C'est le cas de l'exemple présenté dans l'article d'un robot apprenant à brouiller des oeufs. Obtenir une fonction de récompense appréciant la qualité des oeufs brouillés par le robot de façon optimale et basée sur ses capteurs semble très compliqué. On peut penser à utiliser une fonction simplifiée afin de quand même utiliser de l'apprentissage par renforcement, par exemple en se basant sur la couleur et la température des oeufs. Le problème en procédant de la sorte est alors que le robot va chercher à optimiser cette fonction de récompense simplifiée et que le résultat pourrait mener à un comportement non optimal pour la tâche qu'on souhaite que le robot effectue en réalité.

1.3 Diverses solution proposées

Pour palier à ce manque de fonction de récompense bien définie, les auteurs évoquent tout d'abord l'apprentissage par renforcement inverse pour estimer une fonction de récompense à partir d'exemples de la tâche à accomplir, ainsi que l'apprentissage par imitation, qui vise à reproduire directement un comportement fourni en exemple.

Ces deux approches ne sont cependant pas réalisables facilement lorsque la tâche en elle même est complexe à réaliser car elles nécessitent d'avoir des exemples compréhensibles par l'agent de bon résultats de la tâche à effectuer. Pour le cas de l'exemple présenté dans l'article du robot brouilleur d'oeufs, cela semble ainsi difficilement envisageable: lui proposer un exemple du comportement attendu présupposerait par exemple de connaître les différents paramètres du robot nécessaires pour obtenir des oeufs brouillés, alors que c'est pourtant ce que l'on cherche à optimiser.

1.4 Solution retenue

La solution étudiée par les auteurs est alors de se baser sur des retours fournis par un observateur humain sur les performances de l'agent afin d'estimer une fonction de récompense. Il s'agit ainsi d'utiliser les retours utilisateur à la place d'une fonction de récompense.

Le challenge est alors de réduire le plus possible le nombre d'appel à l'observateur humain afin que l'entraînement reste faisable en un temps raisonnable tout en préservant autant que possible la convergence de la politique vers une politique optimale. En effet, il est inenvisageable de demander à un observateur d'évaluer la performances de toutes les itérations successives de l'agent afin que sa politique converge et il faut alors une méthode permettant un nombre d'appel réduit à l'opérateur.

L'article propose alors de remplacer la fonction de récompense par une fonction de récompense estimée, que l'on met à jour de temps en temps à l'aide d'un

[1] comme proposé par chess.com

observateur humain. Le fonctionnement global de la méthode proposée est alors résumé par la figure 2, étendant le schéma 1 vu précédemment.

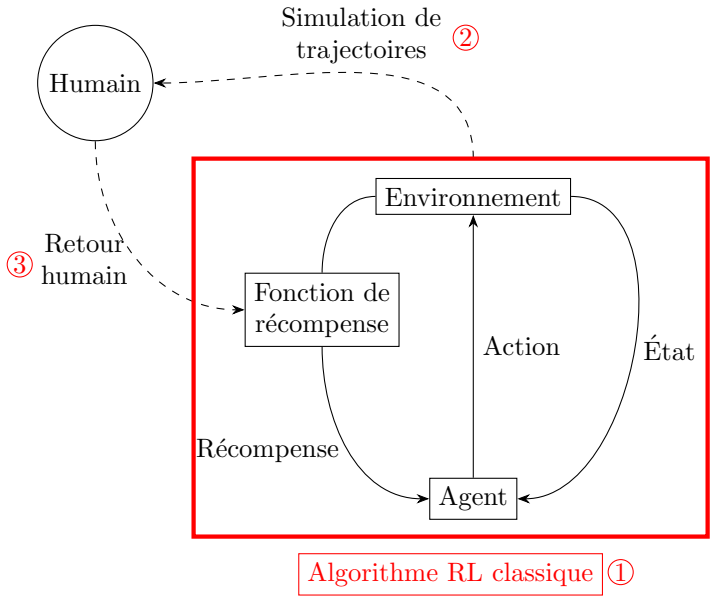


Figure 2: Schéma de fonctionnement de l'apprentissage par renforcement avec retours humains

2 Présentation du modèle

2.1 Cadre mathématique

On considère toujours, comme dans le cours, un agent pouvant interagir avec un environnement:

Pour cela l'agent reçoit des observations $o_t \in \mathcal{O}$ de l'environnent et il y répond par une action $a_t \in \mathcal{A}$ où $\mathcal{O}$ et $\mathcal{A}$ correspondent respectivement à l'ensemble des observations et à l'ensemble des actions possibles. Le choix de l'action a effectuer par l'agent se fait par le biais d'une politique $\pi : \mathcal{O} \to \mathcal{A}$, qui renvoie une action dépendant de l'observation reçue.

On parlera par la suite de trajectoire pour désigner une suite $((o_t, a_t))_{0 \leq t \leq k-1} \in (\mathcal{O} \times \mathcal{A})^k$ d'observations et d'actions.

Classiquement, l'agent recevait également une récompense $r_t \in \mathbb{R}$, correspondant approximativement à la performance de l'action effectuée. En apprentissage par renforcement classique, on cherchait à obtenir une politique maximisant le retour total, une somme plus ou moins ajustée dépendant de ces r_t.

Dans le cadre proposé par les auteurs, on remplace désormais cette récompense par un observateur humain, capable de choisir parmi deux trajectoires celle qu'il

préfère et l'objectif de l'apprentissage est alors de produire une politique permettant de produire des trajectoires que cet observateur humain préfère. Cela se fera ainsi en optimisant itérativement une fonction de récompense estimée, que l'on note $\hat{r}$.

2.2 Description de la méthode

Comme dans le cadre classique d'apprentissage par renforcement, on va considérer une politique π, qui est mise à jour par itérations successives. Cependant, à la différence de ce cadre classique, la méthode proposée tient également à mettre à jour itérativement la fonction de récompense estimée $\hat{r}$. π et $\hat{r}$ sont dans cet article tout deux paramétrisés par des réseaux de neurones profonds.

L'algorithme présente ainsi 3 étapes, représentées sur la figure 2, qui sont répétées dans l'ordre suivant:

1. Optimisation de la politique
2. Sélection et comparaison de couples de trajectoires par un opérateur humain.
3. Optimisation de la fonction de récompense selon les préférences exprimés par le comparateur humain.

Les 3 sous parties suivantes s'attellent à détailler le fonctionnement de ces 3 étapes.

2.2.1 Optimisation de la politique

Cette étape est une application d'un apprentissage par renforcement classique. L'agent interagit avec l'environnement, ie. avec la fonction de récompense $\hat{r}$, fixe tout au long de cette étape et cherche à obtenir une politique optimale vis à vis de $\hat{r}$.

Les auteurs alertent sur le fait que la fonction $\hat{r}$ définie par réseau de neurone profond peut ne pas être stationnaire et qu'il faut ainsi être particulièrement soucieux du choix de la méthode d'optimisation de la politique. Ils recommandent ainsi de renormaliser les récompenses fournies par la fonction de récompense estimée pour que les récompenses restent comparables pour la politique.

Cela se comprend car, sans cette précaution, on peut imaginer qu'après une optimisations de $\hat{r}$, toutes les trajectoires aient désormais une récompense par exemple deux fois plus faible en moyenne qu'avant. Cela impacterait ainsi la vitesse de convergence dans l'étape 1. La renormalisation permet ainsi au contraire de conserver le même ordre de grandeur de récompense au fil des itérations, et donc une vitesse identique de convergence vers une politique optimale à chaque étape 1.

2.2.2 Comparaison de trajectoires

Cette étape consiste à sélectionner deux trajectoires, simulées selon la politique actuelle, optimisée lors de l'étape 1, et à les montrer à l'observateur, généralement par deux petits clips vidéos. L'observateur indique alors alors la trajectoire qu'il préfère ou bien que les trajectoires sont équivalentes ou incomparables.

Le résultat retourné par cette étape est alors une liste de triplets $(\sigma^1, \sigma^2, \mu)$ avec σ^1, σ^2 deux trajectoires et μ, une distribution sur $\{1, 2\}$ indiquant la trajectoire préférée.

Si le comparateur arrive à choisir parmi les deux une trajectoire qui lui parait la meilleure, la distribution charge uniquement cette trajectoire tandis que si les deux trajectoires lui semblent équivalentes, la distribution est uniforme sur ces deux trajectoires.

Un point clé est la manière dont sont choisis les couples de trajectoires: l'article recommande ainsi de choisir en priorité des trajectoires dont la variance des récompenses prédites est la plus élevée, ce qui peut se comprendre dans le fait qu'il y a peu d'intérêt à choisir des trajectoires dont on est relativement sur de la performance et qu'il vaut mieux explorer les trajectoires à l'issue incertaine.

2.2.3 Optimisation de la fonction de récompense

On note désormais pour toutes trajectoires σ^1, σ^2, $\sigma^1 \succ \sigma^2$ si l'utilisateur préfère la trajectoire σ^1 à la trajectoire σ^2.

L'objectif est de s'approcher d'une fonction de récompense r telle que

$$\sigma^1 \succ \sigma^2 \Leftrightarrow \sum_{i=0}^{k-1} r(o_i^1, a_i^1) > \sum_{i=0}^{k-1} r(o_i^2, a_i^2).$$

Pour faire cela, on interprète $\hat{r}$ comme un prédicateur de la préférence du comparateur humain dans le sens suivant :

$$\hat{P}(\sigma^1 \succ \sigma^2) = \frac{\exp\left(\sum \hat{r}(o_t^1, a_t^1)\right)}{\exp\left(\sum \hat{r}(o_t^1, a_t^1)\right) + \exp\left(\sum \hat{r}(o_t^2, a_t^2)\right)} \tag{1}$$

On considère alors une entropie-croisée entre les prédictions effectuées par $\hat{r}$ et les préférences exprimées par le comparateur humain.

Pour un triplet $(\sigma^1, \sigma^2, \mu)$ constitué de deux trajectoires et de la préférence exprimée par l'utilisateur, cela s'écrit

$$-\mu(1)\log(\hat{P}(\sigma^1 \succ \sigma^2)) - \mu(2)\log(\hat{P}(\sigma^2 \succ \sigma^1)).$$

L'article enjoint alors à minimiser l'entropie croisée totale,qui correspond à la somme sur tout les couples comparés de l'expression précédente, ie

$$loss(\hat{r}) = -\sum_{(\sigma^1, \sigma^2, \mu)} \mu(1)\log(\hat{P}(\sigma^1 \succ \sigma^2)) + \mu(2)\log(\hat{P}(\sigma^2 \succ \sigma^1). \tag{2}$$

Pour un simple triplet $(\sigma^1, \sigma^2, \mu)$, on remarque que minimiser cette expression selon $\hat{P}$, revient à choisir

$$\hat{P}(\sigma^1 \succ \sigma^2) = \begin{cases} 1 & \text{si } \mu(1) = 1 \\ 0 & \text{si } \mu(2) = 1 \\ 1/2 & \text{si } \mu(1) = \mu(2) = 1/2 \end{cases} \tag{3}$$

et donc à obtenir une probabilité s'ajustant exactement au choix de l'utilisateur.

On peut ainsi avoir l'intuition que minimiser la somme globale permettra effectivement d'obtenir une fonction de récompense permettant de reconstituer les préférences de l'utilisateur, et ce d'autant plus que le nombre de comparaisons effectuées par l'humain et le nombre de trajectoires visitées seront grand.

Plus concrètement, la fonction de récompense est initialisée de manière aléatoire, et on peut par exemple effectuer une descente de gradient sur les paramètres du réseau de neurones, où chaque information donnée par l'utilisateur entraîne la fonction de récompense dans la direction souhaitée.

Cette approche est analogue au calcul du classement Elo, utilisé notamment aux échecs, où le score de deux joueurs avant la rencontre permet de prédire une probabilité de victoire de chacun des deux joueurs et où le score après la rencontre est d'autant plus mis à jour que le résultat est éloigné du résultat théorique. En effet, si le joueur le mieux classé bat le joueur le moins bien classé, ce n'est guère une surprise et le niveau relatif des joueurs est bien respecté, ce qui ne nécessite alors pas de bousculer le classement. Au contraire, si un joueur que l'on pensait moins bon arrive à battre le meilleur, il faut mettre à jour le classement pour rapprocher le score des deux joueurs, et ce d'autant plus que leur classements initiaux étaient éloignés. Cette même logique s'applique à nos trajectoires: si une trajectoire que la fonction de récompense pensait mauvaise se révèle meilleur qu'une autre que l'on pensait bonne, il faut drastiquement faire évoluer les probabilités, et donc la fonction de récompense ajustée. Au contraire, un résultat peu surprenant devrait faire peu évoluer les probabilités et la fonction de récompense.

3 Analyse des résultats

Les chercheurs ont implémenté cet algorithme dans TensorFlow en s'interfaçant avec MuJoCo et l'environnement d'apprentissage Arcade.

Dans un premier temps, les chercheurs tentent de résoudre une série de tâches de référence pour l'apprentissage par renforcement sans récompense observée : l'agent doit alors apprendre l'objectif de la tâche en utilisant les préférences humaines (un humain doit sélectionner la meilleure de deux trajectoires qu'on lui propose ce qui constitue pour l'agent un retour d'information). Le but étant de résoudre une tâche en un temps raisonnable et avec le moins de requêtes possibles. Cette première série d'expériences montre que les retours humains durent au total entre 30 minutes et 5h.

Afin de mieux apprécier ces résultats et la performance de l'algorithme, deux comparaisons sont faites : la première permet de comparer ces résultats

expérimentaux à ceux obtenus lorsque l'agent ne reçoit pas de retour humain mais la préférence permettant de maximiser la récompense ; la deuxième quant à elle permet de comparer les résultats obtenus avec les résultats de référence en apprentissage profond. Cette démarche vise davantage à obtenir une performance similaire à celle de l'apprentissage profond sans pour autant avoir accès à des informations sur les récompenses et en s'appuyant sur un retour d'information plus même si le retour d'information formulé par des humains a le potentiel pour surpasser l'apprentissage par renforcement classique. C'est ce que nous détaillerons dans la suite du propos.

3.1 Simulations robotiques

Cette étude s'intéresse à 8 tâches de simulation robotique. Les fonctions de récompense associées à ces tâches sont des fonctions linéaires des distances, des positions et des vitesses, et toutes sont une fonction quadratique des features. Parmi ces tâches, une tâche simple a été inclue pour comparer les résultats finaux. L'agent a été entraîné suivant plusieurs conditions d'apprentissage :

- à partir de la récompense réelle
- avec un retour humain (700 requêtes)
- 350 requêtes synthétiques
- 700 requêtes synthétiques
- 1400 requêtes synthétiques

Nous pouvons nous intéresser à la figure ci-dessous détaillant les résultats obtenus en moyennant 5 exécutions de l'algorithme pour les différentes conditions introduites plus haut (exception de l'expérience avec un retour humain pour laquelle l'algorithme a été exécuté une seule fois). Cette figure nous permet d'abord de remarquer que les résultats obtenus pour chaque tâche ne sont pas identiques : les méthodes d'apprentissage par renforcement les plus performantes ne sont pas les mêmes en fonction de la tâche considérée.

Un second constat que nous pouvons faire est que la courbe bleue foncée (1400 requêtes synthétiques) montre que l'algorithme présenté est légèrement plus performant que s'il avait reçu la vraie récompense (courbe orange). Ceci peut s'expliquer par le fait que la fonction de récompense apprise ait été mieux entraînée : la procédure d'apprentissage attribue une récompense positive à tous les comportements qui sont généralement suivis.

On peut également remarquer que les résultats obtenus avec 700 requêtes synthétiques sont assez proches de ceux que l'on obtient avec une récompense réelle.

Par ailleurs, l'apprentissage par renforcement utilisant le retour d'information humain présente des résultats assez fluctuants bien que la tendance soit superposable à celle des autres méthodes décrites.

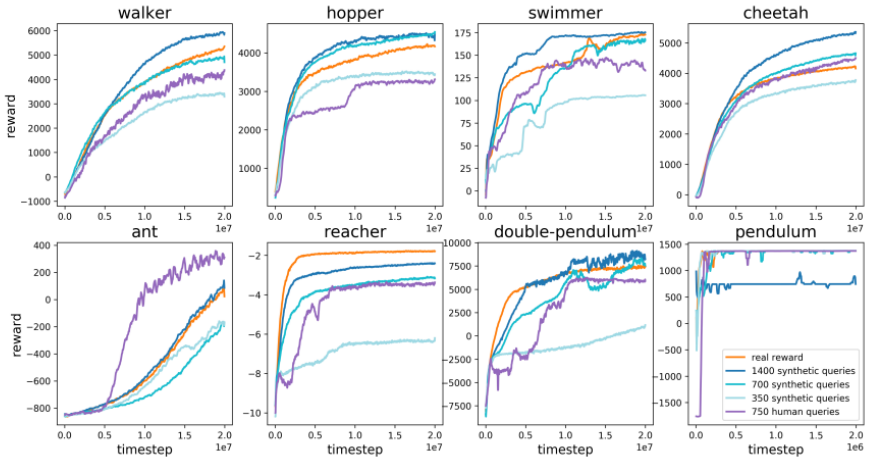


Figure 3: Résultats obtenus pour 8 tâches de simulations robotiques

Enfin, le retour humain s'avère deux fois moins efficace ou tout aussi efficace que la méthode utilisant la récompense réelle. Toutefois, pour une des huit tâches, l'apprentissage par renforcement utilisant le retour humain apparaît comme étant la méthode la plus performante.

3.2 Atari

A présent, l'étude s'intéresse à un ensemble de 7 jeux Atari. On compare à nouveaux différentes méthodes d'apprentissage :

- 5500 requêtes à un évaluateur humain
- 350 requêtes synthétiques
- 700 requêtes synthétiques
- 1400 requêtes synthétiques apprentissage par renforcement classique

Les résultats sont représentés dans la figure ci-dessous :
Ces résultats montrent que la méthode des auteurs présente des performances différentes en fonction de la tâche : le plus souvent cette méthode a une moins bonne performance que l'apprentissage par renforcement classique mais il arrive tout de même que la méthode étudiée présente pour 2 des 7 tâches une performance égale voire meilleure que la méthode classique. Aussi, la vitesse d'apprentissage reste plus élevée pour l'apprentissage par renforcement classique que pour les méthodes utilisant un retour d'information synthétique ou humain en général. De plus, cette figure montre que les performances du retour d'information humain et synthétique sont globalement similaires en considérant le même nombre de requêtes et restent comparables à celles du retour

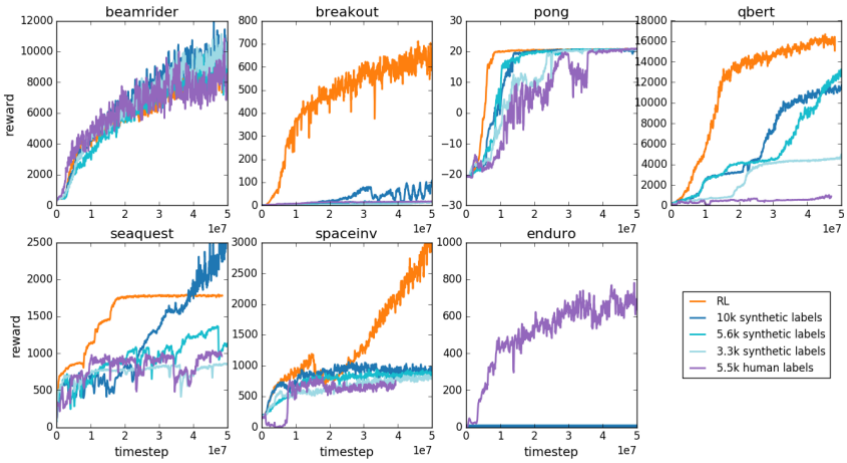


Figure 4: Résultats obtenus pour les 7 jeux Atari étudiés

d'information synthétique avec 40% de requêtes en moins. Ceci peut s'expliquer par : des erreurs humaines dans la réponse à la requête des incohérences entre les différentes personnes fournissant un retour d'information On peut noter que l'algorithme fonctionne le moins bien pour qbert, ceci peut s'expliquer par le fait que les vidéos transmises étaient courtes et déroutantes ; ce qui rendait l'évaluation humaine difficile

3.3 Apprentissage de nouveaux comportements

Les résultats explicités précédemment permettent de comprendre l'efficacité de la méthode. Néanmoins, l'objectif de départ était de pouvoir résoudre des tâches pour lesquelles il n'est pas possible de définir une fonction de récompense.

Cette étude montre alors que l'algorithme implémenté peut apprendre de nouveaux comportements complexes. Par exemple, le robot Hopper apprend en moins d'une heure (après 900 requêtes à l'humain) à faire un saut arrière périlleux en tâchant d'atterrir à la verticale et à répéter cette action comme on peut le voir sur la vidéo Hopper Backflips : lien vidéos Le deuxième exemple évoqué est le robot apprenant à se déplacer en se tenant sur une seule jambe après moins d'une heure d'apprentissage et 800 requêtes. Un troisième comportement a pu être appris de novo : apprendre à une voiture à rester presque au même niveau que les autres voitures bien que l'arrière-plan change souvent. Ce dernier entraînement a nécessité 1300 requêtes.

3.4 Modifications envisagées

Les auteurs proposent certaines modifications dans le but de mieux comprendre les performances de l'algorithme présenté :

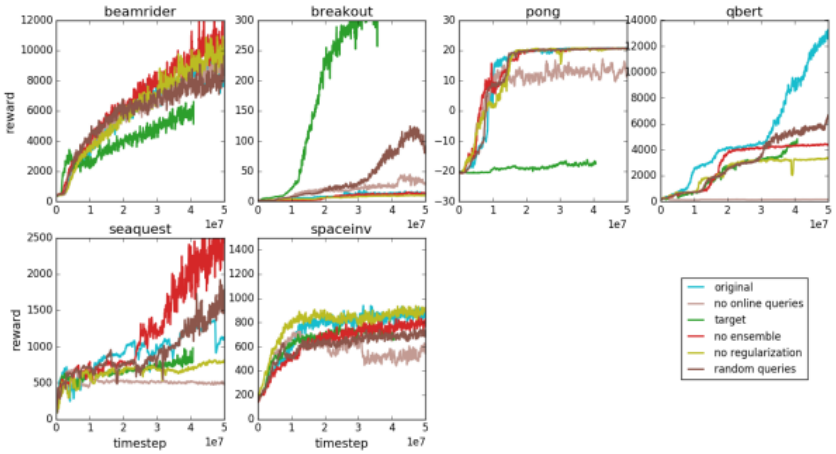


Figure 5: Résultats obtenus avec les modifications proposées pour les jeux Atari

- Choix des requêtes uniformément au hasard plutôt que celles pour lesquelles il y a un désaccord
- Entraînement d'un seul prédicteur plutôt que d'entraîner un ensemble
- Entraînement sur les requêtes collectées au début plutôt que tout au long de l'entraînement
- Suppression de la régularisation
- Pour les tâches de robotique : utilisation de segments de trajectoire de longueur 1
- On considère à présent un oracle fournissant la vraie récompense totale sur une trajectoire et l'estimation de la fonction de récompense est ajustée à ces récompenses totales en utilisant l'erreur quadratique moyenne.

Le résultat de ces modifications est présentée dans la figure ci-dessus :
On constate que l'apprentissage hors ligne du prédicteur de récompense présente une faible performance : la distribution d'occupation n'étant pas stationnaire : le prédicteur ne collectera qu'une partie de la vraie récompense qu'il va chercher à maximiser. Néanmoins, ce processus peut conduire à un comportement atypique ne correspondant pas à celui observé si l'on tenait compte de la vraie récompense (par exemple éviter à l'agent de perdre des points sans en marquer). Cela montre que le retour d'information humain doit aller de pair avec l'apprentissage par renforcement tout au long du processus d'apprentissage. Dans cette étude, les requêtes à l'humain prenaient la forme de comparaisons à faire car il semblait plus facile d'obtenir des comparaisons cohérentes qu'un score absolu cohérent. Les auteurs ont cherché à comprendre comment l'utilisation

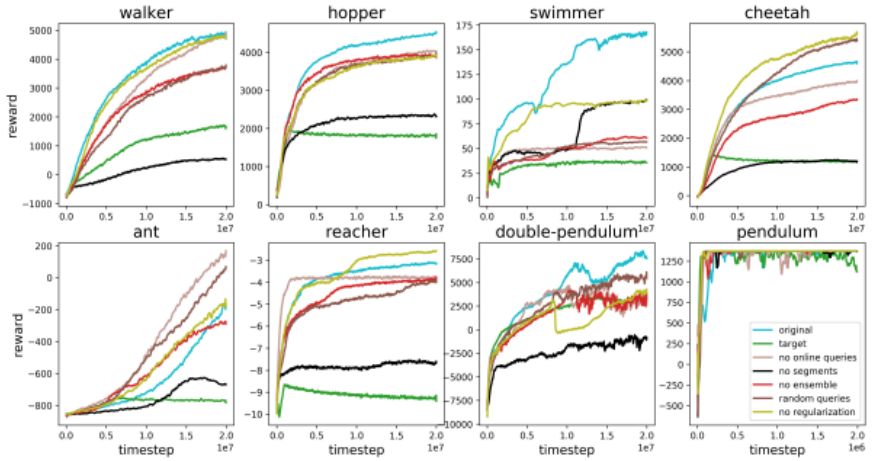


Figure 6: Résultats obtenus avec les modifications proposées pour les simulations robotiques

de comparaisons pouvaient affecter la performance. On constate alors que la prédiction des comparaisons fonctionne mieux que la prédiction de scores. Ceci s'explique par le fait que l'échelle des récompenses varie de façon importante ce qui rend le problème de régression plus difficile. Pour finir, les auteurs ont constaté que fournir des clips vidéos plus longs était significativement plus utile que fournir un clip court : les évaluateurs humains ont souvent besoin d'un temps au départ pour comprendre la situation avant de pouvoir comparer.

3.5 Conclusion et limites identifiées

Alors que les interactions entre l'agent et son environnement sont le plus souvent moins coûteuses en temps d'apprentissage mais aussi en temps humain que les interactions agent-humain, cette étude montre qu'il est possible d'envisager une interaction agent-humain moins complexe et moins coûteuse en temps de calcul. Cet article montre qu'il est bel et bien possible d'implémenter un algorithme d'apprentissage par renforcement sans connaître la fonction de récompense ou dans le cas où la fonction de récompense est difficile à définir.

Cette étude présente une méthode novatrice en matière d'apprentissage de nouveaux comportements : il devient possible d'apprendre des comportements parfois très complexes en un temps raisonnable grâce aux retours humains. Ces conclusions ouvrent la voie à de nombreuses applications pour lesquelles il est difficile de définir une fonction de récompense pour une tâche donnée. En ce sens, cet article incite à des travaux futurs visant à appliquer cette méthode à d'autres types de tâches complexes.

Il aurait été intéressant de demander à un grand nombre de personnes de fournir un retour humain afin de déduire des tendances générales et d'effacer l'effet des erreurs. Cette étude montre que des clips courts et déroutants donnent des résultats peu probants. De ce fait, il pourrait être intéressant d'adapter la qualité et la durée des vidéos fournies de sorte qu'elles fournissent une information suffisante. Aussi, étudier la performance de cette méthode pendant une durée plus longue nous aurait permis de déduire la tendance de l'apprentissage (savoir si un temps plus long est nécessaire pour obtenir une meilleure performance et ainsi de déduire une durée optimale d'apprentissage pour différentes tâches)

4 Application : Résolution d'un labyrinthe

Pour implémenter la méthode, nous voulions programmer un objectif assez simple que l'agent aurait à atteindre. Nous avons choisis comme problème la résolution de labyrinthe car la gestion des actions est suffisamment simple étant donnée qu'il s'agit de trajectoires discrètes. En effet, une direction est définie en chaque temps comme un des quatre élément haut, bas, gauche, droite. Ainsi, les actions correspondant à une trajectoire seront définies comme une des quatre direction du plan.

Nous avons dans un premier temps implémenté la modélisation de labyrinthes ainsi qu'une méthode simplifiée, reprenant l'idée de l'article dans un cadre simplifié. Cette méthode utilise une fonction de récompense simple ayant plusieurs composantes modulé par quelques paramètres que l'on optimise grâce aux retours utilisateurs. Il ne s'agit pas d'un véritable apprentissage par renforcement mais plutôt d'une optimisation de la fonction de récompense afin d'ensuite récupérer successivement des trajectoires de taille finie optimale, et de s'approcher de la résolution du labyrinthe.

Dans un deuxième temps, nous avons implémenté la méthode complète de l'article avec une politique et une fonction de récompense basées sur des réseaux de neurones profonds et donc une optimisation plus générale de la fonction de récompense basée sur les retours de l'utilisateur. Elle implémente un véritable apprentissage par renforcement, qui permet ensuite à l'agent de décider de sa trajectoire case par case.

4.1 Cadre simplifié

4.1.1 Définition du labyrinthe

Le labyrinthe choisi pour cette première méthode est de taille 10 par 10. Il contient une case sortie immobile et un joueur (ou agent) qui se déplace dans ce labyrinthe. Dans un premier temps, on souhaite que l'agent parvienne à résoudre le labyrinthe 7 qui est en ligne droite. Dans un second temps, on souhaite qu'il puisse résoudre n'importe quel labyrinthe. Nous avons donc pour cela implémenté un algorithme de création de labyrinthes aléatoires, dont un un exemple est présenté figure 8.

Sur toutes ces visualisations, le joueur est représenté par le point rouge tandis que la sortie est située au niveau de la croix verte.

4.1.2 Définition des paramètres

Comme défini en 2.1, ne trajectoire est une suite de couple d'états et d'action. Dans cette première étude, l'état o_i est définit par la position du joueur, la position de la sortie, les murs entourant le joueur et l'ensemble des cases précédemment visitées.

La position du joueur pl et de la sortie ex sont des couples de coordonnées. Les murs sont définis comme une fonction $mur(d) = 1$ s'il y a un mur dans la

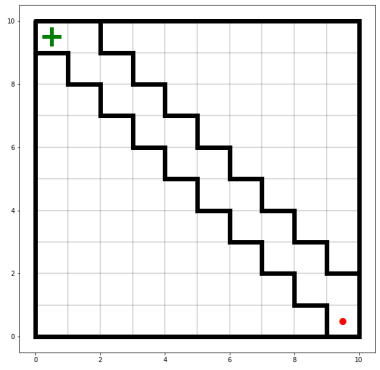


Figure 7: Premier labyrinthe simple

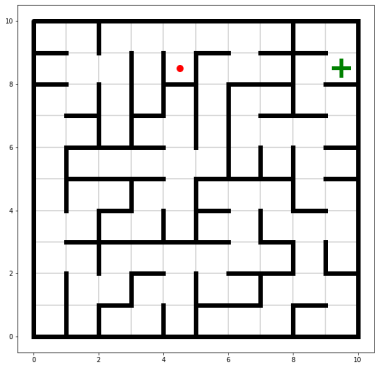


Figure 8: Labyrinthe aléatoire

direction d par rapport au joueur et $mur(d) = 0$ s'il n'y a pas de mur. Les cases précédentes E sont définies comme une liste de coordonnées en deux dimensions.

L'action a_i est définie comme la direction d dans laquelle va aller le joueur.

Ainsi, nous pouvons définir notre trajectoire de taille k comme

$$\sigma = ((o_0, a_0), ..., (o_{k-1}, a_{k-1}))$$

Nous définissons également comme fonction $\hat{r}$:

$$\hat{r} = \left(2\alpha \left| pl - ex \right| + 3\beta \text{mur}(d) - \gamma \sum_{e \in E} \left| e - pl \right| \right) \mathbb{1}_{\{\alpha+\beta+\gamma=1\}} \tag{4}$$

qui sera notre fonction de récompense dont on cherchera à optimiser les paramètres. Cette fonction permet de bien approcher la résolution du labyrinthe avec les trois paramètres α, β et γ :

- Le paramètre α influence l'importance pour l'agent de se rapprocher de la sortie. En effet, minimiser $\hat{r}$ reviendra entre autre à minimiser la distance entre l'agent et la sortie.
- Le paramètre β influence l'importance de se diriger vers une zone accessible. C'est-à-dire de choisir une direction dans laquelle il n'y a pas de mur. En effet, pour minimiser la fonction, il faudra trouver d tel que $\text{mur}(d) = 0$ donc qu'il n'y a pas de mur.
- Le paramètre γ influence l'importance pour l'agent de ne pas repasser plusieurs fois au même endroit. En effet, pour chaque case déjà explorée, les zones de la trajectoire devront être les plus éloignées.
- L'indicatrice permet de ne pas optimiser tous les paramètres à 0.

Comme les trois paramètres du systèmes n'ont pas la même échelle, il a été rajouté des constantes empiriques 2 et 3, qui permettent d'aider la convergence.

4.1.3 Demande des préférence

L'agent a besoin de retours humains afin de converger. Pour ce faire, le programme va s'occuper de récupérer des éventuelles préférences précédemment stockées, puis successivement proposer deux trajectoires différentes et demander à l'utilisateur laquelle des deux est la meilleure.

Après le choix de l'utilisateur, le programme va enregistrer cette nouvelle préférence avec une $\mu = (1, 0)$ ou $\mu = (0, 1)$ selon celle qui est préférée où μ est la densité définie en 2.2.3. L'utilisateur peut également choisir que les deux trajectoires sont équivalentes et le programme va l'enregistrer avec $\mu = (\frac{1}{2}, \frac{1}{2})$. Enfin, l'utilisateur peut décider de ne pas enregistrer la préférence, si les trajectoires sont incomparables par exemple.

Pour choisir les trajectoires à afficher, s'il n'y a pas encore de préférence, le programme va en générer deux aléatoirement. S'il existe au moins une préférence, le programme va utiliser la fonction de scoring. Nous rappelons que nous cherchons à minimiser $\text{loss}(\hat{r})$ définit comme :

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

Ainsi, le programme cherche à déterminer les α, β et γ optimaux pour $\hat{r}$, soit

$$\hat{r}(\sigma) = \min_{(\alpha,\beta,\gamma)\in[0,1]} \text{loss}(\hat{r}).$$

Ainsi, pour chaque trajectoire possible σ^i, le programme va calculer $\hat{r}(\sigma^i)$ et va choisir la trajectoire qui minimise cette fonction. Cette trajectoire est appelée trajectoire optimale.

Enfin, le programme permet, après l'entraînement, de résoudre le labyrinthe. Cette méthode consiste à calculer une trajectoire de taille n et à l'effectuer dans le labyrinthe, puis à recommencer jusqu'à résolution du labyrinthe.

4.1.4 Résultats

Pour les résultats ci dessous, nous avons comparé des trajectoires de taille 4 et résolu le labyrinthe avec des trajectoires de taille 7.

Il est intéressant de remarquer qu'après chaque nouvelle préférence ajoutée, le calcul des paramètres α, β et γ évolue, ce qui montre bien une amélioration de la prédiction. De plus, avec seulement 21 préférences référencées, l'agent est capable de résoudre le labyrinthe 7 mais aussi certains labyrinthes aléatoires 8.

Pour le simple labyrinthe 7, l'agent va directement vers la sortie en ligne droite car il n'y a pas de mur entre elle et la sortie donc le paramètre α le pousse à aller vers la sortie et le paramètre β la pousse à aller de l'avant.

Néanmoins, sur un labyrinthe aléatoire 8, l'agent bloque parfois lorsque la sortie est juste de l'autre côté d'un mur car elle essaie de passer par ce chemin mais elle retourne ensuite en arrière grâce à la mémorisation du chemin parcouru dicté par le paramètre γ. Néanmoins, comme au bout de 7 déplacements l'agent à oublié le chemin parcouru précédemment, il peut se retrouver bloquer dans certains chemins de grande taille.

Il est possible d'augmenter le nombre de déplacement lors du calcul de la meilleur trajectoire mais cela est très coûteux en ressources. En effet, comme un déplacement correspond à un choix parmi 4, cela revient à multiplier par 4 le nombre de trajectoires à chaque élongation de la trajectoire. Le nombre de trajectoire est donc exponentiel.

Une autre méthode pour faciliter la résolution est d'augmenter le nombre d'information dans l'environnement. En effet, il aurait été possible de rajouter la position de tous les murs existants dans le labyrinthe. Cela forcerait l'agent a choisir un chemin ne comportant aucun mur et à voir les choses arriver en avance.

Vous trouverez sur le lien du github différents gifs résolution de labyrinthes avec cette méthode.

4.2 Cadre général

4.2.1 Présentation

Afin d'implémenter la méthode générale issue de l'article, nous nous sommes inspirés de [1], qui l'implémente afin de comprendre le fonctionnement pratique de la méthode et surtout avoir un exemple fonctionnel d'apprentissage par renforcement à adapter à notre résolution de labyrinthe.

Pour évaluer et comparer la performance des différents politiques obtenues sur un même labyrinthe, nous estimons la durée moyenne (en nombre d'actions) pour atteindre la sortie du labyrinthe, D par

$$l(\pi) = \mathbb{E}_\pi \left[\max(D, 100) \right]. \tag{5}$$

afin d'obtenir un score fini, même dans le cas où la politique ne permet pas à l'agent d'atteindre la sortie dans certaines situations.

Pour mesurer l'efficacité de la méthode de l'article, pour des raisons de practicité, nous avons fait le choix de la mettre en pratique sur des labyrinthes de petite taille (3×3). En effet, la convergence de la fonction de récompense estimée demande beaucoup de retour humains lorsque le nombre d'états augmente. L'article parle en effet de centaines de retours afin d'obtenir des résultats et, bien que cela soit très raisonnable pour une application industrielle, ce l'est moins dans le cadre de cette implémentation illustratrice.

4.2.2 Résultats

Nous avons fait tourner la méthode sur deux labyrinthes de taille 3×3, l'un vide (9) et l'autre comportant des murs (10), et donc plus complexe à résoudre, afin notamment de comparer la vitesse d'apprentissage dans ces deux cas.

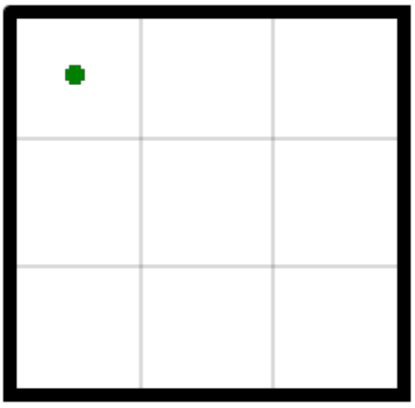


Figure 9: Labyrinthe 3×3 sans murs

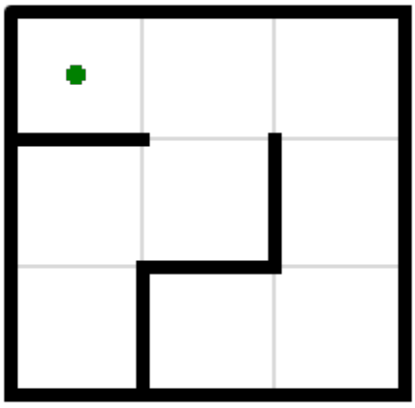


Figure 10: Labyrinthe 3×3 avec murs

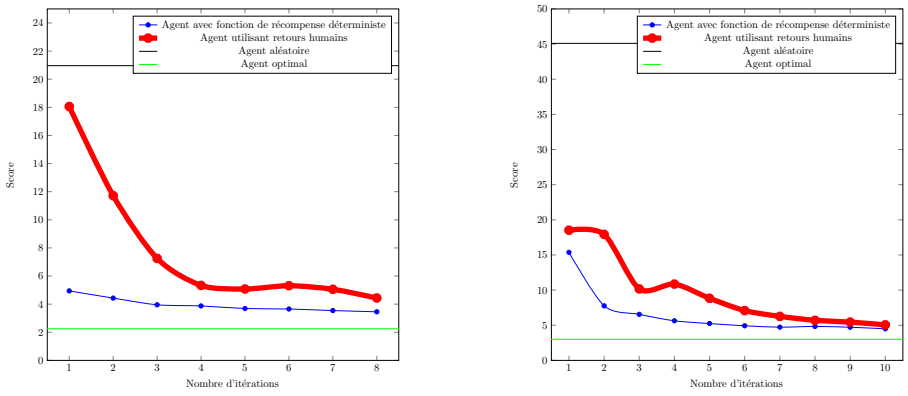
Nous avons également comparé l'agent issu de cet article, s'entraînant avec la fonction de récompense estimée, avec un agent effectuant un apprentissage par renforcement classique, avec comme fonction de récompense la distance entre la sortie et le joueur:

$$r(s) = |s - ex|_1 \tag{6}$$

Les résultats, ci dessous ont été obtenus en considérant les paramètres suivants:

- 10 répétions de la boucle totale.
- 1 000 simulations pour faire converger la politique de l'agent à chaque itération, soit 10 000 au total.
- 5 demandes de comparaisons à chaque itération, soit 50 au total
- Une évaluation des performances à chaque itération, en estimant le score défini par 5 par une moyenne empirique sur 1000 trajectoires.

Par ailleurs, nous avons renormalisé à chaque étape les récompenses obtenues par la fonction de récompense pour obtenir une moyenne de -2 et un écart-type de 2, ce qui permet d'obtenir une récompense moyenne négative, et ainsi éviter au mieux la création de chemin avec une récompense positives (c'est-à-dire de chemins sur lesquels l'agent préfère boucler afin d'amasser des récompenses au lieu d'aller vers la sortie). Cela permet ainsi de privilégier les trajectoires les plus directes vers la sortie.



(a) Performance sur le labyrinte 9

(b) Performance sur le labyrinte 10

Figure 11: Performance de différents agents pour résoudre un labyrinthe

Sur les figure 11a et 11b sont présentées en fonction du nombre d'itérations la performance, au sens défini par 5, de deux agents, l'un entraîné classiquement avec la fonction de récompense 6, l'autre, en rouge sur le graphique, avec la méthode proposée par l'article. Pour l'agent entraîné classiquement, le nombre

d'itération consiste en le nombre d'exécutions de l'étape 1 uniquement sur les 3 étapes de la méthode.

Ces courbes sont présentées pour comparaison avec les performances d'un agent aléatoire, effectuant chaque action uniformément au hasard, et à celles d'un agent optimal, dont on peut facilement définir le comportement dans cet exemple simple.

On remarque ainsi sur le graphique 11a qu'au bout d'une itération, c'est à dire 5 demandes de comparaison, notre agent est déjà meilleur qu'un agent aléatoire. Cela montre ainsi que notre algorithme fonctionne et apprend, car il est meilleur que le hasard. De plus,au bout de quelques itération, les performances des deux agents sont quasiment similaires. Ainsi, dans ce problème simple, un très petit nombre d'informations de la part d'un humain suffit quasiment répliquer les performances de la fonction de récompense choisie.

Sur le graphique 11b, l'allure des courbes est similaire, avec au début l'agent implémentant le modèle de l'article beaucoup moins performant que celui avec la fonction de récompense naïve, puis les courbes se resserrent à mesure que la fonction de récompense estimée apprend des retours humains, jusqu'à faire quasiment aussi bien, en très peu d'itération.

La fonction de récompense étant assez adaptée à ce problème simple, on remarque qu'un agent entraîné avec cette fonction de récompense reste meilleure qu'avec notre fonction de récompense estimée. Cependant, le fait que les courbes soient proches permettent de confirmer l'utilité et la convergence de ce type de méthodes et permet de nourrir l'espoir que cela fonctionne également lorsque la définition de fonction de récompense explicite est moins aisé.

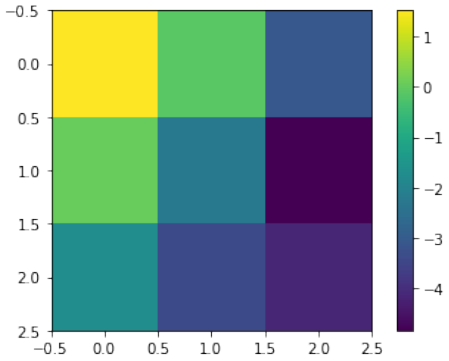


Figure 12: Fonction de récompense optimisée sur le labyrinthe 9

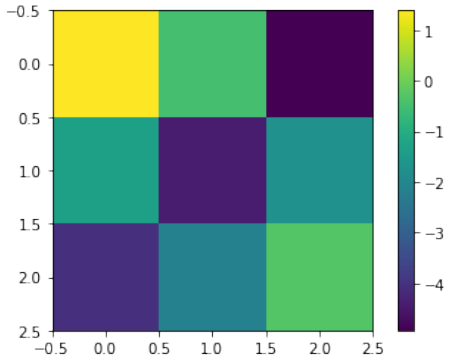


Figure 13: Fonction de récompense optimisée sur le labyrinthe 10

Par ailleurs, les figures 12 et 13, présentant les récompenses obtenues sur les différentes cases du labyrinthe révèlent que la méthode permet bien d'apprendre une fonction de récompense cohérente, une récompense élevée étant par exemple donnée pour les cases les plus proches de la sortie.

4.3 Remarques & Éventail des problèmes rencontrés

4.3.1 Ajuster le taux d'apprentissage de l'algorithme

Le taux d'apprentissage doit être finement ajusté, aussi bien pour l'optimisation de la politique que pour celle de la fonction de récompense, afin que l'algorithme général se passe bien.

Si le taux de convergence de la politique est trop important, il y a un risque que l'agent "hallucine" une politique qui lui semble correcte à partir des première trajectoires issues de la fonction de récompense initiale, aléatoirement choisie et qu'elle se retrouve figée. En effet, s'il y a convergence vers une politique optimale en très peu d'itérations, il est possible que la politique soit désormais quasi-déterministe dans le sens où, pour chaque état, une action a une probabilité de transition extrêmement proche de 1. Si le taux de convergence de la fonction de récompense est trop faible, on risque de se retrouver dans un cadre similaire, où la fonction de récompense restant quasi-similaire entraîne la politique optimale sur une fausse piste.

Si le taux de convergence de la politique est trop faible, comme dans le cadre de n'importe quel problème d'optimisation, il y un risque que la convergence soit excessivement lente.

4.3.2 Des choix compliqués

La méthode implémentée peut parfois demander successivement de faire uniquement des choix entre des trajectoires que l'on peut qualifier de médiocres,comme des trajectoires tentant d'aller dans les murs, voir même demander un choix entre deux trajectoires identiques (même si ce risque diminue avec la taille de l'ensemble des trajectoires). On peut alors se retrouve à ne rencontrer que des mauvais choix.

En effet, il existe un risque de boucle de rétroaction négative: si le comparateur humain voit pendant quelques itérations uniquement des mauvaises trajectoires (en raison de l'aléa initial privilégiant certaines mauvaises trajectoires par exemple), la non comparaison entre une bonne trajectoire et une mauvaise trajectoire ne permet pas d'entraîner la fonction de récompense à reconnaître les trajectoires bonnes des mauvaises mais seulement les très mauvaises des mauvaises. Ainsi cette fonction de récompense peut se mettre à favoriser les trajectoires mauvaises et ce biais empêche la sélection de bonnes trajectoires par la politique qui évolue dans le sens de la fonction de récompense. Ainsi, le comparateur humain ne se voit in fine pas présenter de bonnes trajectoires mais seulement différentes catégories de mauvaises, et la boucle est bouclée.

4.3.3 Valeurs initiales et convergence

Nous avons utilisé des politiques initiales et des fonctions de récompense qui étaient initialisées aléatoirement et il apparaît sur notre implémentation une forte dépendance à cette initialisation. En effet, dans certains des cas, la fonction de récompense initiale favorise des trajectoires défavorables et notre modèle

assez basique ne parvient pas à surmonter cette mauvaise initialisation, si bien que l'on se retrouve de nouveau dans la situation rencontrée dans la sous-section précédents.

On pourrait penser à essayer dans la mesure du possible d'initialiser la politique et la fonction de récompense par des fonctions à priori, afin de partir d'une situation plus ou moins favorable, ou bien considérer des modèles plus complexes, qui sauraient converger même depuis ce genre de minima locaux dans lequel notre modèle simple semble parfois se retrouver.

Des modèles plus robustes et un meilleur choix des trajectoires proposés (ie. pas aléatoire, comme nous l'avons implémenté, mais un choix visant à maximiser la quantité d'informations apprise) pourraient potentiellement permettre de se débarrasser de quelques-unes de ces contraintes rencontrées durant notre exploration de cette méthode.

5 Conclusion

La méthode présentée dans cet article présent un premier pas pour l'entraînement de modèles de machine learning en apportant un apport humain qui ne pourrait être retranscrit dans une fonction de récompense simple. Cette méthode utilise un modèle intermédiaire encodant une fonction de récompense estimée, ce qui permet de réduire drastiquement le nombre d'appel à l'utilisateur et pourrait permettre une extension à des problématiques complexes réelles. Cet article prouve ainsi la faisabilité d'un tel système s'entraînant à partir de retours humains, et ce à un coût équivalent à un entraînement classique, ce qui est essentiel pour de futurs mises en pratique à plus grande échelle.

L'objectif à long terme serait ainsi de pouvoir entraîner des modèles à partir de retours humains aussi simplement qu'à partir de fonctions de récompenses et alors de pouvoir étendre l'éventail des tâches auxquelles l'apprentissage par renforcement est applicable.

5.1 Prolongements possibles

Comme vu en 4.3.2, il peut parfois être compliqué de choisir une préférée parmi les trajectoires proposées et les choix effectués peuvent avoir peu d'influence pour privilégier les trajectoires réellement intéressantes lorsqu'on ne voit que les mauvaises trajectoires.

Afin de surmonter ce problème, une évolution possible de la méthode présentée pourrait être de remplacer la comparaison de trajectoires par une simple notation de la trajectoire sur une certaine échelle subjective. Par ceci, il n'y aurait plus de cas où les décisions effectuées ne servent pas vraiment à améliorer la performance de la fonction de récompense.

Un des problèmes soulevés par ceci est cependant la difficulté de l'humain pour attribuer une note aux trajectoires et il faudrait définir précisément aux humains comparant les trajectoires les attendus de la notation car il est en général bien plus simple de comparer deux trajectoires que de les noter.

Une autre alternative pourrait être un véritable classement de plus de deux trajectoires au lieu d'une comparaison afin de mieux établir une hiérarchie des différents comportements rencontrés. Cela se rapprocherait de l'idée précédente, tout en se débarrassant de la contrainte de suivre une échelle de notation pour évaluer les trajectoires et reposerait comme l'idée de l'article sur l'aide humaine pour converger.

References

[1] Daniel Eid. Deep reinforcement learning from human preferences in tensorflow. `https://www.youtube.com/watch?v=SieMY_-YcvE`, 2022.