Apprentissage séquentiel de compétences via la motivation intrinsèque et l'apprentissage par renforcement

Hedwin Bonnavaud, Arthur Aubret, Laëtitia Matignon

Apprentissage séquentiel de compétences via la motivation intrinsèque et l'apprentissage par renforcement

Rapport de recherche - Juillet 2021

Hedwin Bonnavaud, Arthur Aubret, Laetitia Matignon

Univ Lyon, Universit«e Lyon 1, LIRIS, CNRS, UMR5205 Villeurbanne, F-69622, France

Résumé L'apprentissage de compétence permet à des agents d'apprendre des comportements distincts deux à deux pour chacune d'elles. Cela leur permet d'améliorer leur exploration, et de maximiser plus rapidement leurs récompenses. Cependant, cet apprentissage se fait de manière uniforme. Or, un agent n'a pas toujours accès aux données permettant un tel apprentissage, et cet apprentissage pose également un problème de complexité. Pour le corriger, nous avons développé une méthode permettant d'apprendre des compétences de manière séquentielle, via l'apprentissage par renforcement et les motivation intrinsèque. Au travers de simulations sur un environnement simple, nous avons pu montrer que les compétences apprises étaient plus distinguable que celles apprise de manière uniforme.
Mots-clés: apprentissage par renforcement, apprentissage de compétences, motivations intrinsèques, oublis catastrophique

Abstract. Skill learning allows agents to learn distinct pairwise behaviors for each of them. This allows them to improve their exploration, and to maximize their rewards more quickly. However, this learning is done in a uniform way. However, an agent does not always have access to the data allowing such learning, and this learning also poses a complexity problem. To correct this, we have developed a method to learn skills sequentially, via reinforcement learning and intrinsic motivation. Through simulations on a simple environment, we were able to show that the learned skills were more distinguishable than those learned in a uniform way.

Keywords: reinforcement learning, skills learning, intrinsic motivation, catastrophique forgetting

1 Introduction

En apprentissage par renforcement, un agent va apprendre par essais-erreur en tentant de maximiser les récompenses résultant de ses actions. Ces récompenses

sont définies extrinsèquement[1], selon la tâche que l'on souhaite faire apprendre par l'agent. Cela peut-être la distance parcourue sans accident pour un agent apprenant à conduire, ou une fonction de distance euclidienne lorsqu'on souhaite qu'il atteigne un état spécifique. Cependant, lorsque les récompenses sont trop rares pour l'agent, il risque d'avoir des actions désordonnées. N'étant pas guidé par une récompense qu'il ne trouve pas, l'agent aura du mal à savoir quelle est la meilleur action à faire, dans sa situation de départ, pour maximiser sa récompense. De plus, les comportements qu'il apprend pour réaliser une tâche précise sont difficilement ré-utilisables pour d'autres tâches. En effet, si l'agent apprend à conduire avec comme récompense sa distance parcourue sans accident, le comportement qu'il apprendra sera de beaucoup se déplacer. De ce fait, même s'il aura appris à conduire sans avoir d'accident, il ne pourra pas ré-utiliser cette compétence pour se garer. Il est donc difficile pour lui de réutiliser les comportements qu'il apprend. Une manière d'adresse ce problème est d'apprendre des actions haut niveau, comme *Franchir l'intersection* par exemple, qui est composée d'un grand nombre d'actions de plus bas niveau comme *démarrer* ou *tourner* elles même composées d'actions plus atomiques. Pour inciter l'agent à apprendre des actions de plus haut niveau, des méthodes d'apprentissage de compétences ont été introduites [4,9,8]. Pour cela, l'agent va recevoir des récompenses différentes selon la tâche qui lui est demandé d'effectuer. Les compétences ainsi apprises seront alors distinguables deux à deux. Pour les apprendre, il va donc tenter de les exécuter les unes à la suite des autres, en les choisissant uniformément. Alors qu'il effectue ces compétences, il conserve en mémoire les données représentant les interactions qu'il a eues avec son environnement. Ces données lui permettent, de manière plus globale, de ré-apprendre les compétences qu'il a faites par le passé. Pour cela, l'agent va sélectionner les données uniformément, et va donc ré-apprendre l'ensemble des compétences.

Cet apprentissage uniforme des données pose plusieurs problèmes. Si parmi l'ensemble des compétences certaines sont très bien maîtrisées, l'agent va tout de même les ré-apprendre et va ainsi gaspiller ses capacités de calcul. De plus, plus le nombre de compétences est important, plus l'agent va avoir besoin d'une quantité importante de données pour continuer son apprentissage uniforme. Enfin, cette méthode d'apprentissage impose de conserver en mémoire un grand nombre de données, afin d'éviter le problème du sur-apprentissage[2]. Cependant, cesser d'apprendre de manière uniforme sur ces données, va forcer l'agent à apprendre les compétences dans un ordre précis. Dès lors, le risque est que, au sein des réseaux de neurones du modèle, l'apprentissage d'une nouvelle compétences écrase l'apprentissage des compétences précédemment apprises. Ce problème est connu sous le nom d'*oubli catastrophique*.

Dans ce contexte, la problématique de mon stage était d'étudier la possibilité d'apprendre des compétences de manière séquentielle. Dans un premier temps,

[1] Qui est externe à l'agent.

[2] Aussi appelé *overfitting*. Problème survenant lorsqu'un réseau de neurone apprend par coeur une faible quantité de données ne représentant pas l'ensemble des données auquel il peut-être confronté. Il perd alors en capacité de généralisation.

j'ai donc réalisé une étude de l'état de l'art des méthodes d'apprentissage par renforcement et d'apprentissage de compétences. J'ai ensuite réalisé une analyse des méthodes traitant le problème de l'oubli catastrophique, afin de déterminer lesquelles étaient les plus adaptées à l'apprentissage séquentiel de compétences.

Les méthodes sélectionnées étant classiquement appliquées à des problèmes de classification, elles ont du être adaptées pour les rendre compatibles à notre contexte d'apprentissage par renforcement, puis testées sur des benchmarks classiques d'apprentissage de compétences. L'évaluation de l'approche proposée a été réalisée en comparant les compétences apprises de manière séquentielle et uniforme selon différents critères.

Dans un premier temps, nous étudierons plus en détail les domaines formant le contexte de notre étude, puis nous nous pencherons sur l'état de l'art réalisé. Par la suite nous détaillerons les méthodes sélectionnées pour répondre à notre problématique, avant d'en observer les résultats.

2 Définition et contexte

Notre agent apprend des compétences en combinant motivation intrinsèque et apprentissage par renforcement. Dans cette partie, nous expliquerons le fonctionnement de l'apprentissage par renforcement puis nous définirons la motivation intrinsèque

2.1 Apprentissage par renforcement

En apprentissage par renforcement, un agent [3] interagit avec son environnement par essais-erreurs, en suivant un *processus de décision markovien.*

Le processus de décision markovien (MDP) [30] est une manière de modéliser les interactions qu'un agent a avec son environnement, afin de maximiser l'espérance des récompenses qui en découle. Comme détaillé dans cette revue [3], il est défini par :

- S l'ensemble des états atteignables,
- A l'ensemble des actions possibles,
- T la fonction de transition $T : S \times A \times S \to \mathbb{P}(s_{t+1} = s' | s_t = s, a_t = s)$ avec s_t l'état de l'agent au temps t, a_t l'action prise par l'agent au temps t et s_{t+1} l'état de l'agent au temps $t + 1$,
- R la fonction de récompenses $R : S \times A \times S \to \mathbb{R}$,
- $\gamma \in [0; 1]$ le facteur d'atténuation,
- $\rho_0 : S \to \mathbb{P}(S)$ la distribution initiale d'états.

[3] Terme utilisé pour désigner un individu, un personnage ou un robot, possédant des capteurs (pour capter un état depuis son environnement) et des actionneurs lui permettant d'effectuer des actions (pouvant influencer son état dans l'environnement).

Lorsqu'un agent commence une série d'interactions, il est placé dans un état $s_0 \sim \rho_0(\cdot)$ défini par ρ_0. A partir de cet état, l'agent va choisir une action $a \in A$. Suivant la fonction de transition T, l'action qu'il a effectuée va mener l'agent dans un nouvel état s'. En suivant la fonction de récompense R, une récompense r_t va être donnée à l'agent à partir de l'interaction (s, a, s') tel que $r_t = R(s, a, s')$. A partir de son nouvel état s', l'agent va pouvoir entamer une nouvelle interaction avec son environnement, puis va continuer cette boucle. Un ensemble d'interactions est appelé épisode, et se termine lorsque l'agent atteint un état dit *terminal*.

L'objectif d'un MDP est de pouvoir maximiser la somme des récompenses à long terme définie par :

$$\sum_{t=0}^{\infty} \gamma^t r_t \tag{1}$$

Les algorithmes d'apprentissage par renforcement associent les états aux actions au travers d'une politique $\pi : S \times A \to \mathbb{R}$. Cette politique, qui peut être définie comme étant la stratégie de l'agent, va, à partir d'un état, définir l'action à prendre. L'objectif de l'agent est alors d'apprendre une politique optimale $\pi*$ définie par :

$$\pi^* = arg\, max_\pi \mathbb{E} \sum_{t=0}^{\infty} \gamma^t R_{a_t \sim \pi(\cdot|s_t)}(s_t, a_t, \pi(s_{t+1})) \tag{2}$$

Pour maximiser cette espérance, les algorithmes basés valeur vont apprendre à associer des valeurs aux couples états / actions. Plus une action effectuée dans un état aura une valeur importante, plus elle mènera, selon l'algorithme, vers des récompenses importantes. La manière d'apprendre ces valeurs, et de les utiliser pour trouver la politique optimale varie selon les algorithmes. Par exemple, Deep Q-Network (D.Q.N.)[25] utilise un réseau de neurones qui, à partir d'un état, apprend la valeur associée à chaque action depuis cet état.

Pour choisir l'action à effectuer dans un état, l'agent peut effectuer sa politique courante, ou bien d'explorer son environnement pour en acquérir de nouvelles. Le choix entre exploration et exploitation est défini par la méthode d'exploration de l'algorithme. Cette méthode peut être de tirer une valeur aléatoire, et explorer (prendre une action aléatoire) si cette valeur est inférieur à un seuil ϵ, ou bien exploiter ses connaissances en choisissant l'action qu'il considère être la meilleur sinon, mais d'autres méthodes d'exploration existent.

Lorsque l'espace d'action est continu, on ne peut pas associer une sortie du réseau de neurones à chaque action. Certains algorithmes utilisent donc un réseau de neurones acteur, dont l'entrée est l'état dans lequel l'action doit-être effectuée, et la sortie une valeur continue correspondant à l'action à effectuer et non plus à sa valeur. Pour évaluer les décisions prises par ce réseau, et donc lui permettre d'apprendre, l'algorithme utilise également un réseau de neurones critique qui estime la valeur d'un couple état/action. Cette architecture est connue sous le nom d'*actor critic*. Comme indiqué dans la table 1 parmi les algorithmes utilisant

cette architecture, on distingue 2 catégories d'algorithmes, ceux *on-policy* et ceux *off-policy*. Les algorithmes *on-policy* prennent en compte la stochasticité induite par leur politique dans leur apprentissage. De ce fait, ils ne peuvent donc pas conserver les données de leurs interactions pour les réapprendre plus tard. A l'inverse, les algorithmes off-policy ont une utilisation optimale de leur données, car les données issues des interactions passées peuvent-être réutilisés en étant ré-apprises plus tard.

A3C[24] est un algorithme actor-critic *on-policy*. DDPG [19] est un algorithme utilisant lui aussi une architecture "acteur-critique". Mais est, contrairement à A3C, un algorithme off-policy. Cependant, cet algorithme reste assez instable, et a tendance à surestimer la valeur des états. Pour corriger ses défauts, TD3 [11] ajoute un bruit gaussien aux actions, et utilise deux réseaux critiques pour corriger la surestimation en utilisant le plus pessimiste des deux. SAC [14] utilise un acteur stochastique, dont l'entropie 7 de la politique est utilisée pour évaluer la valeur associée à un état. Plus la politique déduite d'un état est entropique, plus cette état sera considéré comme ayant de la valeur. Cela a un double effet. Le fait que la politique de l'agent soit entropique dans un état donné, signifie que quoi que fasse l'agent dans cet état, il se trouvera dans une bonne posture pour maximiser ses récompenses, et l'état est donc très intéressant pour lui. Cet algorithme se trouve être le plus stable et le plus efficace de tous ceux sur lesquels je me suis penché. C'est donc l'algorithme que nous avons choisi d'utiliser dans nos expérimentations.

Algorithmes On-policy
A3C
Algorithmes Off-policy
DDPG
TD3
SAC

TABLE 1: Algorithmes d'apprentissage par renforcement utilisant une architecture acteur-critique

La politique apprise par un algorithme d'apprentissage par renforcement maximise l'espérance des récompenses reçues de l'environnement. Cette récompense va donc définir le comportement que l'agent aura à la fin de son apprentissage. L'utilisation d'une récompense associée uniquement à un état, limite grandement le champ des comportements que l'agent peut apprendre. Heureusement, des comportements plus complexes peuvent-être appris grâce aux récompenses intrinsèques.

2.2 Apprentissage par motivations intrinsèques

Les récompenses intrinsèques [28,3] ne sont pas données par l'environnement. Elles sont calculées par l'agent, pour chacune de ses interactions, et peuvent donc dépendre de son nouvel état, de son action, ou de son état de départ, mais aussi être fonction de son passé selon ce que l'agent a conservé en mémoire. De cette manière, les récompenses intrinsèques peuvent permettre à l'agent d'apprendre des comportement plus complexes. Par exemple avec une récompense correspondant à $1/(n(s)+1)$ avec $n(s)$ le nombre de fois que l'agent a rencontré un état s, l'agent aura une récompense plus importante lorsqu'il se rend dans des états qu'il a peu rencontrés, et vas alors être incité à explorer son état [16].

Ces récompenses intrinsèques sont indispensable aux algorithme d'apprentissage de compétences que nous allons voir. Afin de comprendre pourquoi les récompenses extrinsèques ne sont pas suffisante, commençons par comprendre quel type de compétences sont apprises et pourquoi cela a un intérêt.

2.3 Apprentissage de compétences

Lorsque l'agent apprend par renforcement, avec des récompenses extrinsèques associés aux états qu'il traverse, ces dernières peuvent-être rares. En effet, il est possible que les états proches de l'état s_0 ne donnent aucune récompense. Dans ce cas, l'agent peinera à atteindre ces récompenses, et ne pourra donc pas apprendre sa politique optimale π^*, car son apprentissage va être guidé par ses récompenses. Les méthodes d'apprentissage de compétences sur lesquelles nous nous sommes penchés[4], visent donc à apprendre des compétences qui sont distinctes deux à deux, et favorisent l'exploration et l'assignation de valeur. En effet, en cherchant à apprendre des compétences qui sont distinctes, l'agent va apprendre des compétences qui se rendent dans des zones différentes de l'espace d'état, et qui vont ainsi s'éloigner du point de départ de l'agent. En les exploitant, l'agent va donc beaucoup explorer, en se rendant dans des zones qu'il n'aurait sûrement jamais atteint avec une politique désordonnée, n'étant pas guidé par des récompenses. De plus, en évaluant ses compétences plutôt que sa politique dans un état, ou bien une action dans un état, l'agent trouvera plus vite sa politique optimale car il évaluera alors des actions de plus haut niveau plus pertinentes pour réaliser sa tâche (pour sortir de la pièce, il est plus simple de choisir entre *atteindre la porte* et *atteindre la clef* une seul fois plutôt que de choisir à chaque seconde entre *avancer la jambe gauche* et *avancer la jambe droite*. Cette attribution de valeur, plus connu sous l'anglicisme *credit assignment* sera donc amélioré. Ces avantages sont illustrés par la figure 1

Avant de nous pencher sur les différents algorithmes d'apprentissage de compétence composant l'état de l'art, nous allons nous pencher sur le fonctionnement en détail de l'algorithme DIAYN [9], afin de comprendre comment un agent peut apprendre des compétences, sans que celles-ci lui soit dictés extrinsèquement. Par la suite, nous utiliserons cet algorithme comme point de référence pour

[4] Il existe également des algorithmes apprenant d'autres types de compétences.

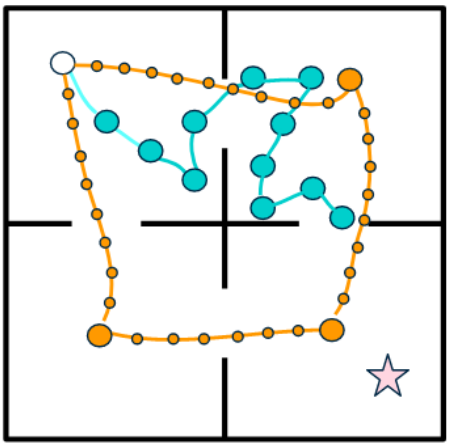

FIGURE 1: Différents comportements attendus par une politique exploitant des compétences (en jaune) et par une politique plus classique (en cyan) dans un environnement vaste présentant une récompense éparse. Ici, les compétences apprises permettent à l'agent de se rapprocher plus facilement de la récompense, tout en accélérant l'association de valeur. Dans cet environnement, le seul état à donner une récompense sera donné par l'étoile.

comprendre d'autres algorithmes d'apprentissage de compétences, en observant les différences entre eux.

Pour maximiser l'espérance de la somme des récompense intrinsèque qu'il génère à chaque interaction, l'agent utilise un algorithme d'apprentissage par renforcement. En apprentissage de compétence, la politique, qui définit les actions prises par l'agent, doit-être différente d'une compétence à l'autre. En effet, nous souhaitons que les comportements que l'agent associe à chaque compétences soit différents pour chaque compétence. Or, pour choisir sa politique, l'algorithme d'apprentissage par renforcement utilise l'état qu'il reçoit de l'environnement. Puisque la politique doit alors dépendre à la fois de l'état et de la compétence, les deux sont concaténés pour former un état intrinsèque, qui est donné à l'algorithme d'apprentissage par renforcement lui offrant toutes les informations nécessaires pour choisir une politique pertinente.

Au début de chaque épisode, une compétence $c \in [0; C]$ (avec C le nombre de compétences total à apprendre) est choisie puis concaténée à l'état s_0 pour former un état intrinsèque si_0. L'algorithme d'apprentissage par renforcement va alors pouvoir effectuer sa première interaction en choisissant une action $a_0 \sim \pi(si_0)$. En suivant la fonction de transition T du MDP, l'environnement va donner à l'agent un nouvel état s'. Maintenant qu'une interaction est terminée, l'agent va donc devoir se générer une récompense. L'objectif de DIAYN est d'apprendre des compétences distinctes deux à deux. Pour cela, l'algorithme apprend à se rendre dans des zones distinctes de l'espace d'états pour chaque compétences. Il vas

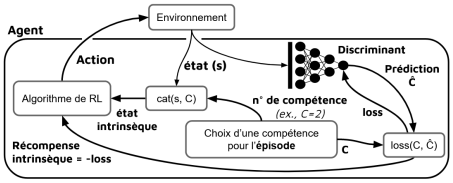

FIGURE 2: schéma représentant le fonctionnement interne de l'algorithme DIAYN

donc chercher à se rendre dans des états qui à eux seuls, permettent de deviner la compétence qui était effectuée par l'algorithme d'apprentissage par renforcement. L'algorithme va donc maximiser l'information mutuelle entre les états et la compétence exécutée. 7 Pour générer sa récompense intrinsèque, l'agent utilise donc un second module qui est chargé de deviner la compétence qui est effectuée par l'agent, uniquement à partir du nouvel état s' capté de l'environnement. Pour cette tâche, DIAYN utilise un réseau de neurones nommé discriminant. Ce réseau prend en entrée l'état s' capté de l'environnement, et effectue une prédiction de la compétence $\hat{c}$, avec une couche de sortie comprenant un neurone pour chaque compétence, donnant la probabilité que l'état soit associé à chacune d'elle. En comparant la prédiction $\hat{c}$ et un *one-hot encoder* représentant la compétence c qui était effectuée par l'agent, l'agent va pouvoir générer une erreur e utilisée pour entraîner le discriminant, mais également une récompense intrinsèque $ri = -e$. Plus globalement, le discriminant vas alors agir comme un classifieur, en attribuant des labels **intrinsèques** aux états.

L'architecture de l'algorithme DIAYN est détaillée dans la figure 2. Dans cette figure, la compétence 2 est sélectionné pour l'exemple.

Au début de l'apprentissage, la politique de l'agent sera la même pour chaque compétences. Cependant, sa stratégie d'exploration va l'amener pour certaines compétences à se rendre dans des zones légèrement différentes. L'apprentissage parallèle du discriminant vas alors l'amener à associer ces nouvelles zones à la compétence qu'il effectuait au moment de l'exploration, et va ainsi donner une récompense positive à l'agent lorsqu'il se rend dans cette zone pour la compétence en question. Petit à petit, la politique de l'agent vas être amenée à se discriminer pour chaque compétences. La figure 3 montre des exemples de compétences apprises par l'algorithme DIAYN.

Le contexte de notre domaine d'application, va nous servir de base pour comprendre les méthodes de l'état de l'art. Nous pouvons désormais nous pencher sur ces algorithmes.

3 État de l'art

L'objectif du stage étant d'apprendre des compétences de manière séquentielle, il était nécessaire de connaître les méthodes d'apprentissage de compétences,

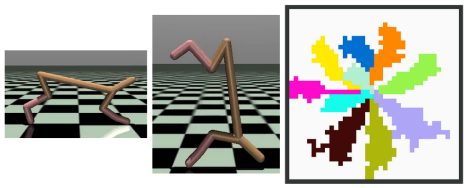

FIGURE 3: Différentes compétences apprises par DIAYN. A gauche, deux compétences apprises dans l'environnement Half-chetah de Mujoco, à droite 10 compétences apprises dans un environnement *gridworld* simple, où l'état de l'agent correspond à ses coordonées, et les actions possibles sont les déplacement dans les 4 directions discrète. Les états par lesquels sont passés l'agent sont coloriés en fonction de la compétence qu'il effectuait au moment où il s'y est rendu. Pour obtenir cette représentation, chaque compétences a été exécutée 5 fois.

mais aussi les méthodes nous permettant de corriger le problème de l'oubli catastrophique. Dans un premier temps, nous allons donc nous pencher sur l'état de l'art des méthodes d'apprentissage de compétences, avant d'analyser les différentes méthodes traitant l'oubli catastrophique. Cela nous permettra par la suite d'étudier lesquelles de ces méthodes sont les plus compatibles avec notre problématique.

3.1 L'apprentissage de compétences

Maintenant que nous en savons plus sur l'algorithme DIAYN 2.3, nous pouvons nous intéresser à d'autres algorithmes de l'état de l'art.

La manière dont cette récompense est calculée, peut varier d'un algorithme à l'autre. Le module chargé de reconnaître la compétence, peut-être un réseau de neurone, appelé dans ce cas discriminant (comme c'est le cas pour DIAYN), ou bien un VAE [5]. Comme nous avons vu, DIAYN utilise un discriminant simple servant à classifier directement les états par lesquels passe l'agent en compétence. L'agent a donc une récompense intrinsèque associée à chacune de ses actions. ELSIM [4] utilise plusieurs discriminant et plusieurs algorithmes d'apprentissage par renforcement pour former une arborescence dans la classification des états en compétences. Cette arborescence va permettre d'effectuer des compétences plus générales et d'autres plus précises. VIC [12] utilise un discriminant pour calculer la récompense intrinsèque. A la différence de DIAYN, cet algorithme ne calcule de récompense qu'au niveau de l'état final de chaque épisode. VALOR [1] utilise un discriminant pour décoder les compétences à partir des trajectoires effectuées par l'agent lors d'un épisode. Pour décoder les trajectoires, une architecture LSTM [15] bi-directionnelle est utilisée dans le décodeur, afin que chaque état de la

[5] variational auto-encoder, un auto-encodeur avec comme espace latent une couche de deux neurones donnant une moyenne et un écart type permettant d'échantillonner les données.

trajectoire ai la même importance. Ainsi, les compétences apprises par VALOR peuvent mener l'agent dans la même direction sans pour autant qu'il effectue la même trajectoire et donc la même compétence. RIG [26] utilise un VAE pour encoder les états. L'espace latent est utilisé comme compétence, et permet de mesurer la distance entre deux états sans utiliser d'oracle[6]. De cette manière, RIG apprend à atteindre des états but. Ainsi, RIG diffère des autres approches de notre état de l'art en apprenant un ensemble de compétences de taille similaire à l'espace d'état de son environnement. EDL [5] utilise un auto-encodeur pour maximiser l'information mutuelle entre les états et les compétences et entraîner le décodeur à la manière de RIG, mais utilise l'information mutuelle entre les états et la compétence pour calculer la récompense intrinsèque à la manière de DIAYN.

	objectif	VAE ou discriminant
DIAYN	$max \sum I(s,c)$	$discriminant(s > \hat{c})$
VIC	$maxI(S_f, C)$	$discriminant(S_f > \hat{c})$
VALOR	$maxI(\tau, c)$	$Dcodeur(\tau < \hat{c})$
RIG	$min(c - c_f)$	$VAE(s > c < \hat{s})$
EDL	$max \sum I(s,c)$	
ELSIM		$discriminant(s > \hat{c})$

Table 2: Tableau comparatif des algorithmes d'apprentissage de compétences

La table 2 récapitule l'ensemble de ces méthodes. La colonne objectif indique ce que l'algorithme cherche à maximiser ou minimiser selon ce qui est indiqué, et correspond donc à une simplification de ce qui est optimisé par l'algorithme. La colonne VAE ou discriminant, indique si l'algorithme utilise un discriminant, un VAE ou simplement un décodeur[7]. Dans ce tableau, s correspond à l'état dans

[6] Connaissance globale de l'environnement. L'agent doit s'en passer autant que possible puisqu'il n'y a pas accès dans des applications réels.

[7] dans VALOR, le discriminant est appelé décodeur par comparaison aux algorithmes utilisant des VAE. Il est précisé dans l'article que c'est l'algorithme d'apprentissage par renforcement et le MDP qui fait office d'encodeur, en générant la trajectoire qui est utilisée comme l'espace latent d'un VAE.

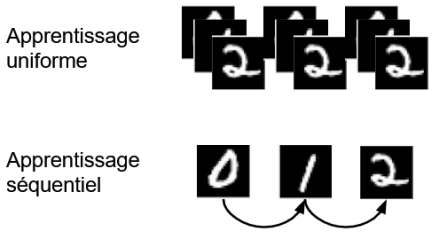

FIGURE 4: Exemple d'apprentissage séquentiel pour un problème de classification. Lorsque le réseau de neurones apprend à classifier des images de 2, il oublie ce qu'il avait appris en apprenant à classifier des images de 1 ou de 0.

lequel se trouve l'agent au moment où la récompense intrinsèque est calculée. $\hat{c}$ correspond à la compétence donnée par le discriminant. S_f correspond à l'état finale, atteint par l'agent à la fin de son épisode. c_f correspond à l'espace latent encodé à partir de S_f. $I > O$ représente un réseau de neurones avec comme entrée I et comme sortie O, et $I > L < O$ représente un VAE encodant son entrée I en un espace latent L, décodé en une sortir O. $I(X, Y)$ correspond à l'information mutuelle 7 entre X et Y, deux variable aléatoires.

3.2 L'oubli catastrophique

En apprentissage profond, lorsqu'un réseau de neurones apprend une tâche[8] (par exemple associer des images à un label prédéfini en classification), ses poids sont modifiés pour tendre vers les poids optimaux pour cette tâche. Si par la suite ce même réseau de neurones apprend une nouvelle tâche, différente de la précédente, ses poids seront modifiés pour tendre vers les poids optimaux de la nouvelle tâche. Cette nouvelle modification des poids va effacer l'action de modification des poids du premier apprentissage (Cf. figure 4) . La capacité du réseau de neurones à effectuer la première tâche va donc se dégrader. Ce problème est appelé problème de l'oubli catastrophique [23].

Pour le corriger, les gradients appliqués au réseau de neurones sont généralement calculés à partir d'une erreur moyenne sur un ensemble de données, appelé batch, contenant des données de plusieurs tâches différentes. Le gradient appliqué au réseau de neurones fera tendre ses poids vers des connaissances globales pour l'ensemble des tâches. Dans ce cas, le réseau de neurones apprend de manière uniforme. Cependant, dans certaines situations, apprendre des connaissances de manière uniforme pose un problème de complexité en temps et en espace. Certains algorithmes d'apprentissage de compétences doivent conserver en mémoire beaucoup de données pour pouvoir ré-apprendre uniformément. De plus apprendre uniformément peut pousser l'agent à ré-apprendre des compétences déjà maîtrisées (puisqu'il les apprend toutes) gaspillant ainsi ses capaci-

[8] ensemble de une ou plusieurs classes ou compétences

tés de calcul. Enfin, apprendre un grand nombre de compétences implique des batch de données plus important pour mieux généraliser les compétences, ce qui va avoir un impact important sur la complexité de l'algorithme. Des méthodes existent pour corriger l'oubli lors d'un apprentissage séquentiel, mais la protection des données passées se fait souvent au dépend de l'apprentissage des nouvelles, problème connu sous le nom du dilemme de la plasticité-stabilité [13].

Il existe plusieurs méthodes pour prévenir l'oubli catastrophique. Lorsqu'on peut se permettre un apprentissage uniforme, c'est généralement cette option qui est utilisée. Cependant, dans certains contextes, apprendre de manière séquentielle peut s'avérer plus avantageux. Dans ce cas, il est indispensable de trouver d'autres méthodes pour contrer l'oubli catastrophique. L'objectif de ces méthodes est de conserver les connaissances apprises par un réseau lors d'un nouvel apprentissage. En plus de l'étude de leur complexité en temps et en espace, ces méthodes se comparent entre-elles en mesurant leur plasticité, et leur stabilité, étant donné que ces approches s'exposent au dilemme de la plasticité-stabilité. Nous définirons donc la plasticité d'une méthode comme sa capacité à apprendre de nouvelles connaissances, et sa stabilité comme étant sa capacité à ne pas oublier ses anciennes connaissances. Les méthodes de correction de l'oubli catastrophique que j'ai donc étudié se divisent en trois familles principales.

Les approches d'isolation 3 visent à isoler des parties[9] du réseau de neurones, et à les conserver pour chaque tâche apprise par le réseau. Ces approches souffrent généralement d'un problème de complexité en espace ou d'un problème de plasticité. Progressive Neural Network (P.N.N.) [27] crée un réseau de neurones pour chaque tâche à laquelle elle est confrontée. Cette méthode ajoute des connections transversales d'un réseau de neurones à l'autre, ce qui lui permet de faire du transfert d'information[10]. Cette méthode soufre d'une mauvaise complexité en espace car elle a besoin de trop de neurones et de trop de connections pour être efficace. PathNet [10] utilise un seul réseau de neurones, et va entraîner un algorithme génétique pour trouver le meilleur chemin associé à chaque tâche dans ce réseau. Cette méthode souffre d'un problème de plasticité car des ressources neuronales sont limitées lorsque le nombre de connaissances à apprendre augmente, mais aussi d'un problème de complexité en temps car ses algorithmes génétiques prennent du temps à être entraînés. PackNet [21] réserve un sous-ensemble de son réseau de neurones pour chaque tâche, ce qui lui pose un problème de plasticité majeur. Notre comparaison de ces approches est représenté dans le tableau 3

Les approches de régularisation 4 sont des approches visant à conserver les anciennes connaissances en réduisant les gradients associés aux nouvelles tâches.

[9] poids, neurones, couches, réseau

[10] Lorsque les connaissances associée à une tâche influences les connaissances d'une autre tâche. On parle de transfert positif lorsque ce transfert augmente le taux de réussite associé à une tâche, et de transfert négatif dans le cas contraire

Approches d'isolation				
	Complexité		Plasticité	Stabilité
	Temps	Espace		
PNN	**	*	***	***
PathNet	**	***	*	***
Packnet	**	***	*	***

TABLE 3: Tableau comparatif des algorithmes de correction de l'oubli catastrophique de la famille des approches d'isolation. Les couleurs représente les performances de l'algorithme dans chaque domaine, allant du rouge (très mauvais) au vert (très bon).

Elles sont généralement peu coûteuses en temps de calcul. EWC [17] (Elastic Weight Consolidation) régule l'erreur en fonction de l'importance moyenne des poids pour les anciennes tâches. Cette méthode simple a un défaut de plasticité et de stabilité, mais a l'avantage de n'avoir besoin de conserver aucune donnée, et de demander peu de ressources de calcul. WVA [18] (Weight Value Atténuation) s'inspire de EWC en réduisant le gradient cette fois-ci directement au niveau de chaque poids en fonction de l'activation qui le franchit. Cette méthode a un important problème de complexité en espace car il a pour cela besoin de conserver les valeurs optimales pour chaque poids et pour chaque tâche. Weight Friction [20] s'inspire du phénomène de friction pour conserver les poids associés à d'anciennes tâches. Plus un poids est important (et qu'il est donc probablement important pour une ancienne tâche) plus son apprentissage est restreint. A-GEM [6] fait tendre le gradient de la nouvelle tâche vers le gradient correspondant à l'entraînement sur des mémoires épisodiques correspondant aux anciennes tâches. Cette méthode est très efficace, mais aussi très coûteuse en temps de calcul, puisque les gradients associés aux mémoires épisodiques doivent-être recalculés à chaque nouvel apprentissage. Notre comparaison de ces approches est représenté dans le tableau 4

Les approches de répétition 5 corrigent l'oubli en réapprenant les anciennes tâches. La manière d'obtenir des données sur lesquelles ré-apprendre peut cependant varier. *Experience replay* [7] , est un algorithme simple qui propose de conserver un sous-ensemble des données observées pendant l'apprentissage d'une classe dans une mémoire annexe. Pour conserver ces données, les auteurs pro-

Approches de régularisation				
	Complexité		Plasticité	Stabilité
	Temps	Espace		
EWC	***	***	**	**
WVA	***	*	**	**
Weight Friction	***	***	**	**
A-Gem	*	**	**	***

TABLE 4: Tableau comparatif des algorithmes de correction de l'oubli catastrophique de la famille des approches de régularisations. Les couleurs représente les performances de l'algorithme dans chaque domaine, allant du rouge (très mauvais) au vert (très bon).

posent d'utiliser l'algorithme réservoir sampling [31]. Cet algorithme permet de sélectionner m éléments dans une liste de N autres éléments, de manière à ce que chaque élément ait la même chance d'être pris, sans connaître le nombre de données N, et en ne parcourant qu'une seul fois cette liste. Cet algorithme est utilisé par *experience replay* pour conserver un sous-ensemble des données associés aux tâches passées dans une mémoire annexe, pour pouvoir les réapprendre plus tard. Pseudo-répétition [2] utilise deux réseaux de neurones pour générer des données à partir de bruit. L'un des réseaux sert de mémoire à long terme, l'autre de mémoire à court terme. A la différence de toues les autres méthodes, CURIOUS [8] corrige l'oubli catastrophique en apprenant des compétences. En sélectionnant les compétences qui seront représentées dans ses batch, il ne les apprend pas toutes uniformément, et est donc soumis au problème de l'apprentissage séquentiel. Pour sélectionner les compétences qu'il apprend, il utilise la valeur absolue du progrès d'apprentissage [11] associé à chacune d'elles. Lorsque cette valeur est nulle, c'est donc que l'agent n'a plus rien à apprendre sur la tâche en question. Il n'a donc pas besoins de la réapprendre. En revanche, si cette valeur (absolue) est positive c'est que la tâche est en train d'être oubliée ou en cours d'apprentissage. Il est donc nécessaire de la ré-apprendre, cependant, cette méthode ne répond pas directement à notre problématique car elle est appliqué à

[11] Évolution du taux de réussite de l'agent sur une tâche donnée. Un progrès d'apprentissage négatif témoigne d'un oublis, un progrès d'apprentissage positif témoigne d'un apprentissage.

des compétences extrinsèques (les compétences sont définis par le développeur, avec une fonctions de récompense propre à chacune d'elle). Combined replay [29], couple les qualités de *Experience replay* et de Pseudo-répétition en générant des données non-pas à partir de bruit mais à partir de données réelles. Le générateur de données va également réutiliser les données générées pour générer d'autres données à partir d'elles, augmentant ainsi la généralisation des données. De plus, les données étant générées par le réseau de neurone utilisé pour la classification, elles sont associées à un vecteur de prédiction lorsqu'elles sont utilisées pour générer une nouvelle donnée. Ce vecteur de prédiction permet de décrire la donnée utilisée, en indiquant de quelle classe cette donnée contient des informations. Notre comparaison de ces approches est représenté dans le tableau 5

Approches de répétition				
	Complexité		Plasticité	Stabilité
	Temps	Espace		
Experience replay	*	*	***	***
CURIOUS	**	*	**	***
Pseudo Répétition	**	***	***	**
Combined Replay	**	**	***	***

TABLE 5: Tableau comparatif des algorithmes de correction de l'oubli catastrophique de la famille des approches de répétition. Les couleurs représente les performances de l'algorithme dans chaque domaine, allant du rouge (très mauvais) au vert (très bon).

Cette analyse nous permet de prendre du recul sur les approches de résolutions d'oubli catastrophique. Comme nous allons le voir certaines sont plus efficaces que d'autres, mais ce n'est pas là le seul critère pouvant nous permettre de choisir la méthode à appliquer à notre apprentissage de compétences.

4 Contributions

La plus part des méthodes corrigeant l'oubli sont appliqués à des problèmes de classification. Les différences entre ce type de problèmes et le notre, vas nous amener à nous restreindre à un sous-ensemble d'entre elles. Par la suite, nous pourrons donc étudier comment ces méthodes peuvent-être adaptés à l'apprentissage de compétences.

4.1 Choix des approches de correction de l'oubli catastrophique

Il existe différentes manières d'apprendre séquentiellement. Les séquences[12] d'apprentissage peuvent contenir plusieurs compétences à apprendre ou une seule. On appelle Task Incremental Learning (TIL) l'apprentissage séquentiel par tâches (plusieurs compétences par séquences) et Class Incremental Learning (CIL) l'apprentissage séquentiel par classe (une seule compétence par séquence).

Les approches de régularisation et d'isolation apprennent en multi-head. C'est à dire que les réseaux de neurones utilisés ont des couches de sorties indépendantes et associées à chaque tâches. Pour les approches d'isolations, ce sont les sous-ensembles du réseau de neurones, ou chacun des réseaux de neurones dans le cas de PNN, qui sont considérés comme des têtes. Lors de l'apprentissage, ces méthodes utilisent l'identifiant de la tâche pour savoir quelle tête utiliser. En apprentissage supervisé, cette information est disponible pendant la phase d'entraînement, car les données d'entraînement sont labelisées et que l'identifiant de la tâche peut-être retrouvé à partir du label de la donnée. Cependant, cela constitue tout de même une information extrinsèque qui n'est pas accessible en phase d'exploitation sur des données réels qui ne sont donc pas labelisées. En apprentissage de compétences, la compétence à effectuer est choisie par l'agent. L'identifiant de la tâche est donc toujours accessible, puisqu'il est concaténé à l'état qui est donné à l'algorithme d'apprentissage par renforcement. Cependant, utiliser plusieurs têtes pose un autre problème. Quelle que soit la tâche en cours d'apprentissage, le discriminant peut recevoir comme entrée n'importe quel état de l'espace d'état. Si l'agent s'est rendu dans une zone de l'espace d'états lorsqu'il effectuait une certaine compétence au début de son apprentissage, il peut aussi bien se rendre dans une zone complètement différente pour la même compétence à la fin de son apprentissage. Si le discriminant utilise donc plusieurs têtes, chacune va calculer une classification de l'espace d'état qui lui est propre. Ainsi, deux compétences d'une même tâche seront très distinctes, alors que deux compétences de deux tâches différences n'auront aucune influence l'une sur l'autre, et pourront ne pas être distinguables. Un tel algorithme utilisant des tâches de n compétences, sera donc identique à un apprentissage uniforme de n compétences seulement. Pour cette raison, l'apprentissage séquentiel de compétences que nous réaliserons se fera en CIL.

[12] un apprentissage séquentiel est composé de plusieurs séquence. Lorsqu'un agent apprend les compétences d'une séquence, il ne peut pas ré-apprendre les compétences des anciennes séquences, sauf si l'algorithme utilisé conserve des données dans une mémoire.

L'utilisation du multi-head est indispensable pour les approches de régularisation et d'isolation [22] sans quoi elles ne permettent plus de corriger l'oubli catastrophique.

Souhaitant donc apprendre des compétences séquentiellement et en mode CIL pour les raisons cités précédemment, j'ai décidé de me concentrer sur les approches de répétition pour corriger l'oubli au sein des réseaux de neurones qui composent l'algorithme d'apprentissage de compétences. Ces méthodes seront appliquées à l'algorithme DIAYN (2.3) [9].

Pour faire de l'apprentissage de compétences en mode CIL, il est nécessaire d'apprendre au moins deux compétences de manière uniforme durant la première séquence, afin d'éviter que le discriminant apprenne a classifier une seule compétence quel que soit l'état en entrée, et n'apprenne donc pas.

4.2 Utilisation de experience replay

L'algorithme *experience replay* 3.2 [7] (E.R.), permet de contrer l'oubli catastrophique de manière simple et efficace. En effet, ré-apprendre les compétences rencontrées via des données conservées dans une mémoire fixe est suffisant pour ne pas oublier les anciennes compétences. Cependant, cet algorithme ne permet pas de résoudre les problèmes de complexité en temps et en espace liés au ré-apprentissage uniforme des batch, et au stockage des données d'interactions dans le buffer. J'ai donc implémenté cette méthode pour nous servir de baseline.

Appliquer cet algorithme à DIAYN Peut se faire de manière assez intuitive, étant donné que DIAYN apprend déjà sur des batch de données sélectionnés aléatoirement dans des petits buffers. En version uniforme, ces buffers conservent les dernières interactions de l'agent. En CIL, ces buffers ne contiennent donc que des données associées à l'unique compétence qui est en train d'être apprise. Pour ajouter des données des anciennes compétences à ses batch d'apprentissage, il est donc nécessaire d'utiliser une mémoire qu'on appellera épisodique (par comparaison à la mémoire épisodique humaine, et non pas par rapport aux épisodes définis par le MDP). A la fin d'une séquence d'apprentissage, une partie des données associée à la compétence qui a été apprise sont transférées dans cette mémoire épisodique. Afin de s'assurer que les données des autres compétences ne sont pas remplacées par ces nouvelles données, la mémoire épisodique est composée d'espaces réservés à chaque compétences. Ces espaces sont d'une taille prédéfinie (200 dans nos simulation) mais sont trop petit pour contenir toutes les données du buffer. Pour sélectionner les données à conserver, l'article d'*experience Replay* conseil d'utiliser la méthode *reservoir sampling*[13] [31]. Cette gestion des données s'applique à deux niveau. D'une part au niveau du buffer de données de l'algorithme d'apprentissage par renforcement, qui a donc un buffer et une mémoire épisodique qui lui sont propre, et d'autre part au niveau du discriminant qui dispose donc également de sa propre mémoire épisodique, et de son propre buffer pour l'apprentissage de la compétence courante.

[13] Méthode permettant de sélectionner n éléments parmi une liste, avec une probabilité d'être sélectionnée égale pour chacun de ses éléments. Cette méthode fonctionne sans connaître la taille de la liste de départ et ne la parcours qu'une seul fois.

4.3 Utilisation de combined replay

Combined replay est un algorithme très proche de *experience replay*, mais est plus efficace lorsque le nombre de données est très faible. en revanche, si l'agent n'est pas limité en mémoire, *experience replay* est plus efficace. Heureusement pour nous, DIAYN est un algorithme qui utilise très peu de données. Alors que mon application de *experience replay* fonctionnait correctement lorsque je lui attribuait un espace de 200 données par compétences, combined replay devrait pouvoir nous permettre d'en utiliser beaucoup moins. C'est principalement pour éviter le sur-apprentissage que nous ne pouvons pas nous permettre d'apprendre sur peu de données. Or, la génération de données réalisé par combined replay, nous permet à partir d'un faible nombre de données, d'en générer de nouvelles. Toutes ces nouvelles données se généralisent de plus en plus en même temps que leur nombre de ré-génération augmentent. Ces données générales, qui non seulement sont générales au sein d'un classe, sont aussi générale au niveau de l'ensemble des données, et sont associées à un vecteur de prédiction donné par le classifieur qui associe cette donnée à l'ensemble des classes. De plus, combined replay est extrêmement compatible avec DIAYN. En effet, son discriminant est lui-même un classifieur, et peut donc être utilisé de la même manière, en générant de nouveau états et en les associant à plusieurs compétences de manière simultanées. Pour que l'algorithme d'apprentissage par renforcement continu d'apprendre, il a besoin qu'une récompense intrinsèque soit associé à ses actions, et a donc besoins que l'environnement lui attribue un nouvel état lorsqu'une action est choisie. Le réseau de neurones choisissant la politique ne peut donc pas appliquer cette méthode de génération de données pour réapprendre des connaissance globales. En effet, l'acteur doit choisir une actions à partir d'un état réel, pour que l'environnement puisse lui attribuer un nouvel état à la suite de son action. Pour corriger l'oubli au niveau de la politique de l'agent, nous avons donc pensé à ré-utiliser le vecteur de compétences généré par le discriminant lorsqu'il prédit la compétence associée à un état. Le fait que le discriminant associe une probabilité importante à chacun des états signifie que cet état est proche des compétences prédite. Cela signifie que l'algorithme d'apprentissage par renforcement peut corriger son oubli en réapprenant la classification des états du discriminant, puisqu'elle contient des informations pertinentes sur l'ensemble des compétences, mais également qu'elle peut apprendre à effectuer plusieurs compétences en utilisant le vecteur de compétence donné par la prédiction du discriminant. En effet, si le discriminant hésite entre deux compétences pour un état, c'est que ces compétences sont proches et qu'il est possible d'apprendre une combinaison de ces compétences.

Une fois ces méthodes construite, nous avons cherché à les appliquer dans des simulation pour les comparer à l'apprentissage uniforme.

5 Résultats

Pour comprendre la manière dont nos simulations ont été réalisées, nous allons dans un premier temps comprendre l'environnement de simulation et la méthode

d'évaluation, avant de nous pencher sur leur résultats, et comment ces derniers ont fait avancer notre méthode.

5.1 Benchmark

De manière générale, les articles présentant les résultats des méthodes dont nous avons parlés tests leur algorithmes dans des environnement similaires les uns des autres. Les environnements les plus utilisés sont sûrement ceux de OpenAI-Gym `https://gym.openai.com/envs`. Cependant, ces environnements ont des espaces d'états vaste. Dans notre cas d'application, lorsqu'un agent apprend à se rendre dans des espace d'action distincts, il est donc difficile de visualiser les compétences de l'agent, et si elles sont bel et bien distinctes. J'ai donc créer un environnement simple, avec un espace d'état en deux dimension. Cet environnement correspond à une pièce contenant 40×40 positions, et dans laquelle l'agent peut se déplacer en observant sa position dans la pièce. Créer cet environnement m'a également permis d'ajouter des fonctions pour générer des représentation des compétences, et d'autres visualisations. Lorsque l'environnement n'est pas précisé, c'est dans celui-ci que sont effectués les simulations.

5.2 Méthode d'évaluation

Comme détaillé précédemment, l'oubli catastrophique surviens lorsque des gradients de taches différentes sont rétro-propagés dans un réseau de neurones. Ainsi, lorsque notre agent apprend de manière séquentielle, ce problème peut survenir sur chacun de ses réseaux de neurones.

Afin de cibler les parties de l'algorithme souffrant de ce problème, nous avons cherché à tester séparément l'algorithme d'apprentissage par renforcement et le discriminant.

Pour cela, une copie du réseau de neurones acteur (donnant la politique) et du réseau discriminant était faite à chaque fin de séquence, après l'apprentissage d'une nouvelle compétence. Ainsi, lorsque nous souhaitions tester le discriminant courant sur sa capacité à reconnaître les anciennes compétences, il nous suffisait de demander aux réseaux de neurones acteur d'effectuer tour à tour la compétence qu'il avaient apprise, et de demander au discriminant de le reconnaître. Inversement, nous pouvions tester la politique courante sur sa capacité à effectuer les anciennes compétences en lui demandant d'effectuer ces compétences, pis en utilisant les anciens discriminants pour les évaluer.

Cependant, nous avons pu observer au fil de plusieurs simulations que cette évaluation posait problème. Lorsque l'agent oublis, son discriminant vas être entraîné à associer tous les états à la compétence courante. Ainsi, en phase de test, la politique semblera ne jamais se tromper, et semblera meilleure que les méthodes corrigeant l'oubli, car les anciens discriminant reconnaîtrons toujours la compétence pour laquelle ils ont été entraîné, quelque soit la politique effectuée par l'agent.

Pour corriger ce problème, nous avons opté pour un mode d'évaluation plus simple. Pour comparer différents algorithmes, nous observons l'évolution de la

motivation intrinsèque de l'agent lorsqu'il effectue l'ensemble des compétences passées. Ainsi, si le réseau de neurones acteur oublie les anciennes compétences et effectue toujours la même politique, le discriminant ne pourra pas distinguer les différentes compétences, qu'il ai lui aussi oublié ou non, et ne pourra donc pas offrir une bonne récompense intrinsèque. De la même manière, si le discriminant oublie, il ne pourra pas reconnaître une compétence effectuée par l'acteur. Lorsqu'il oublie, le discriminant apprend à réduire son erreur en étant très entropique (il associe une probabilité proche à chaque compétence), ainsi, en ne prenant pas de risque, il parvient à associer une probabilité minimale à la compétence réelle. Cependant, il ne peut pas de cette manière, générer une récompense intrinsèque aussi importante que pour un discriminant n'oubliant pas. Enfin, une récompense intrinsèque indique que les compétences sont distinguable, et donc que l'algorithme fait bel et bien ce qu'on attend de lui.

Pour s'assurer de ne pas tomber dans des cas particuliers, nous effectuons sur chacune de nos simulations, plusieurs exécutions (5 en général, le nombre exact pour chacune d'elles est détaillé en annexe) et affichons sur nos graphique le résultats moyens des tests de chacune de ces exécutions, avec une zone coloriée transparente correspondant à la moyenne +/− l'écart type.

5.3 Tests de experience replay appliqué à DIAYN

Nous avons donc voulu observer les performances de cette méthode en comparant différentes implémentations, en faisant varier la proportion de données de la mémoire épisodique et de données associées à la compétence en cours d'apprentissage. Cette proportion peut-être fixe tout au long de l'apprentissage , ou être *proportionnelle*, c'est à dire que la proportion de données issue de la mémoire épisodique est proportionnelle au nombre de compétences qui y sont représentés par rapport au nombre de compétences en cours d'apprentissage. Dans la même simulation, nous avons comparé ces implémentations à DIAYN sans *experience replay*, d'une part apprenant de manière uniforme, et d'autre part apprenant de manière séquentielle, sans aucune stratégie permettant de corriger l'oubli. Les paramètres détaillés de cette simulation sont donnés en annexe .

Les résultats de cette simulations sont représentés dans la figure 5.

Sur ce graphique, nous avons choisi d'observer l'évolution de la motivation intrinsèque de l'agent (axe des ordonnées) au fil des séquences d'apprentissage (axe des abscisses). Il est important de noter que la version de DIAYN apprenant de manière uniforme ("DIAYN UNIFORM") n'a donc qu'une seul séquence d'apprentissage correspondant à l'apprentissage de l'ensemble des données. Elle ne présente donc qu'une seule évaluation de sa motivation intrinsèque sur l'ensemble des données, et sa courbe correspond donc à une ligne horizontale correspondant à la valeur mesurée lors de cette évaluation.

Toutes les mesures ont été faites sur 5 exécutions pour chaque implémentations, et les courbes du graphique représentes les moyennes de ces exécutions. L'aire transparente correspond à la moyenne +/− l'écart type.

En observant ces résultats, nous avons dans un premier temps pensé que les implémentations utilisant *experience replay* souffraient d'un oubli catastro-

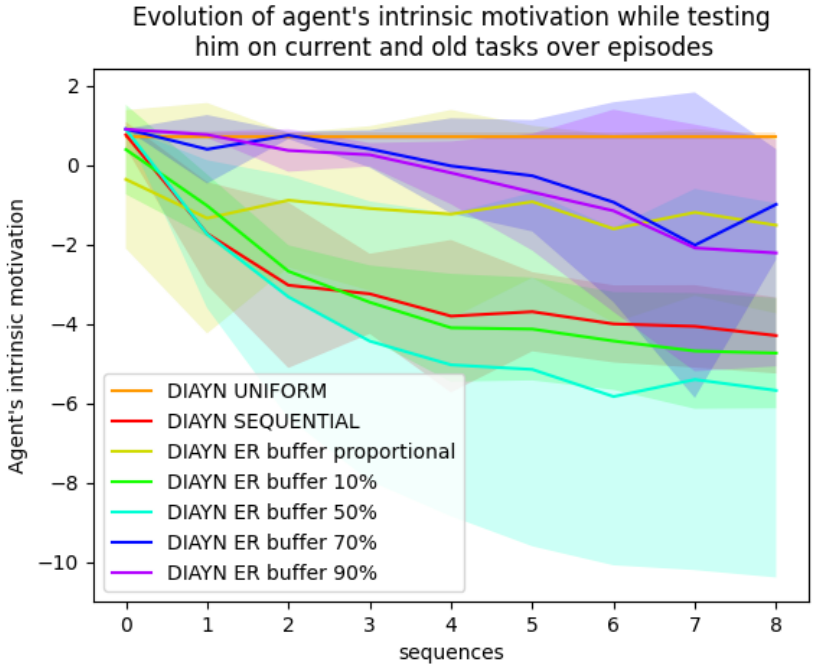

FIGURE 5: Évolution de la motivation intrinsèque générée par l'agent lorsqu'il est testé sur ses anciennes compétences et celles qu'il est entrain d'apprendre, à la fin de différentes séquences d'apprentissage.

phique. En effet, l'agent parvient à s'assurer une bonne récompense intrinsèque lors des premières séquences, puis sa motivation intrinsèque sur l'ensemble des compétences (comprenant donc les compétences passées) baisse au fur et à mesure qu'il se confronte à de nouvelles compétences. Ce comportement étant typique d'un oublis catastrophique.

Pour confirmer cette analyse, nous avons tracé d'une part la disposition des données dans la mémoire épisodique, et d'autre part la disposition des états traversés par l'agent lorsqu'il effectue ses compétences. Sur ces représentations, nous avons colorié les états en fonction de la compétence à laquelle ils étaient associés pour le premier cas, et la compétences qui était exécutée par l'agent lorsqu'ils ont été traversés dans le second cas (figure 6).

Cependant, en comparant les représentations ainsi générés deux à deux, nous avons pu observer que l'exécution des compétences apprises par le passé par l'agent, reste fidèle aux données dans la mémoire épisodique. L'oubli n'était donc pas le facteur qui donnait ce comportement à l'apprentissage. De plus, en observant la disposition des états conservés dans la mémoire épisodique, nous avons pu nous rendre compte que lorsqu'une nouvelle compétence est apprise,

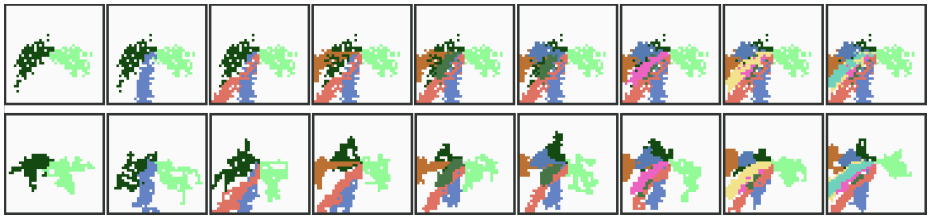

FIGURE 6: Représentation des compétences apprises par l'agent utilisant E.R. De gauche à droite, les représentations à la fin des séquences de 0 (gauche) à 8 (droite). En haut, les états coloriés représentent les états associés mémorisés dans la mémoire des compétences passées, et en bas les états coloriés représentent les états par lesquels est passé l'agent lorsqu'on lui a demandé d'effectuer 5 fois chaque compétence.

ce sont souvent des zones de l'espace d'états (C.A.D. des zones de l'environnement ici, car l'espace d'états correspond aux coordonnées de l'agent) étant déjà associés à d'anciennes compétences qui son associés à la nouvelle. Or, il est plus compliqué pour le discriminant de deviner la compétence effectuée par l'agent si il se rend dans des zones similaires pour plusieurs compétences. Nous en avons donc conclue que l'agent cherchant trop à retourner dans la même zone, empêche le discriminant de retrouver la compétence effectuée. Ainsi, plus les séquences d'apprentissage passent, plus le nombre de compétences se rendant dans la même zone augmentent, et plus la récompense intrinsèque de l'agent est faible. Pour corriger ce problème, nous avons cherché à inciter l'agent à se rendre dans des zones de son espace d'état différente des zones associées à ses anciennes compétences.

5.4 Évolution de l'algorithme utilisant experience replay

Pour cela, une première idée était de limiter l'influence des anciennes compétences sur la nouvelle en incitant l'agent à oublier partiellement ses anciennes compétences lorsqu'il commence à en apprendre une nouvelle. J'ai donc modifié le taux de données associées aux anciennes compétences qu'il réapprend (T_c).

$$T_c = \frac{t}{1 + e^{-x*s+r}} \tag{3}$$

Avec t la proportion ciblée d'anciennes données dans le batch (selon ses paramètres, Cf. figure 5), qui sera atteinte sur le long terme, s la vitesse avec laquelle l'agent va se mettre à réapprendre l'ensemble des compétences passées, et r le retard (en épisode) avant qu'il se mette à ajouter des données associées aux anciennes compétences dans son batch.

Avec cette méthode, une partie des données des anciennes compétences sont remplacés par des données associées à la compétence en cours d'apprentissage.

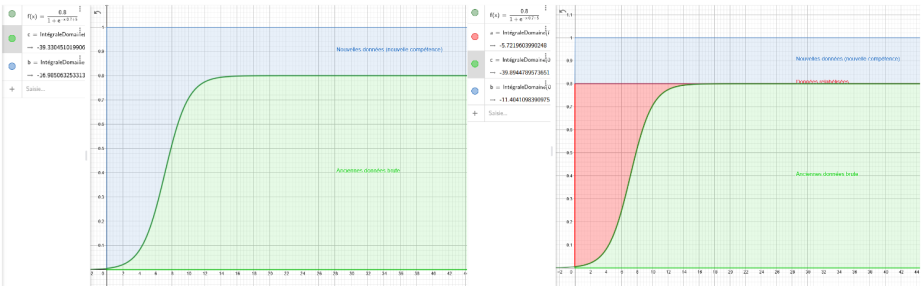

FIGURE 7: Évolution de la provenance des données dans le batch en fonction de l'indice de l'épisode au sein d'une séquence d'apprentissage. A gauche une première version de cette méthode et à droite selon une seconde version utilisant du *rellabelling*. La zone verte correspond aux données provenant de la mémoire épisodique, la zone rouge correspond à des données *relabellisées*, et la zone bleu correspond aux données provenant de l'exécution de la compétence en cours d'apprentissage

J'ai également tenté une seconde méthode, variante de la première, mais cette fois-ci, plutôt que de remplacer les données de la mémoire épisodique par des données associées à la compétence courante, je les ai remplacés par des données re-labélisées. J'ai pour cela utilisé les données conservées dans la mémoire des compétences passées, en remplaçant la compétence associée à ces données par la compétence courante, et la récompense par une récompense arbitrairement mauvaise (-10), ou encore par des récompenses spécialement recalculés par le discriminant. L'origine des données dans le batch en fonction de l'épisode pour ces deux versions est représenté dans la figure 7

Toutes ces variantes n'ont pas permis d'apprendre des compétences plus distinctes, même si elles ont permis à l'agent d'améliorer sa récompense intrinsèque pour qu'elle devienne meilleur que celle obtenu par l'agent DIAYN. Cependant, nous avons modifié la manière de mesurer la motivation intrinsèque de la version uniforme de DIAYN. Au lieux de n'effectuer qu'un seul test à la fin de son apprentissage, nous effectuons des tests à intervalles régulières, en conservant le moment auquel le test a été réalisé. Cela nous permet, en plus d'observer l'évolution de la motivation intrinsèque de DIAYN, au fil de son apprentissage, d'observer la durée de son apprentissage par rapport au implémentations utilisant *experience replay*. Nous avons donc testé ces variantes (nommés "DIAYN ER V2" pour la première et "DIAYN ER V3" pour la seconde, Cf. figure 8). Nous les avons également testés avec différentes proportions ciblées d'anciennes données dans le batch (*70%* et *proportional* comme présentés précédemment), en changeant le paramètre de retard de l'équation **??**lorsqu'il est indiqué "late=r" et avec ou sans délais. L'absence de délais indique que cela correspond à la version initiale de l'application de *experience replay* à DIAYN.

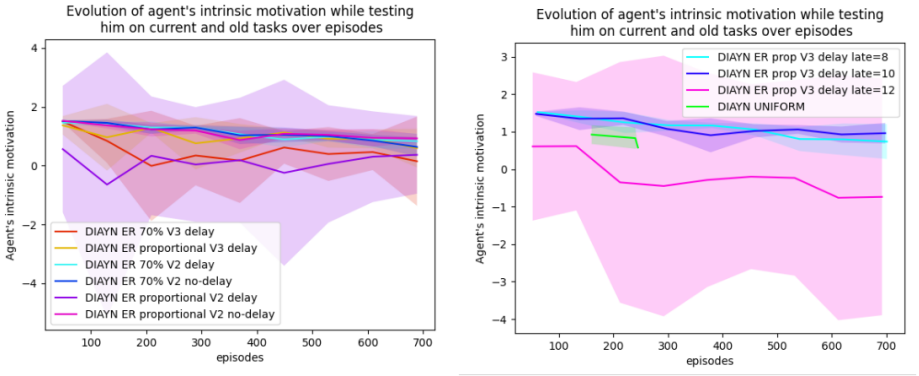

FIGURE 8: Évolution de la motivation intrinsèque de différents agents lorsqu'ils sont testés sur d'anciennes compétences.

Nous pouvons donc remarquer que ces variantes apportent peu d'améliorations par rapport à la version initiale (*no-delay*). Cependant, nous pouvons observer qu'en modifiant les paramètres de la fonctions de délai, retardant l'apprentissage des anciennes compétences, les performances de l'algorithme peuvent-être améliorées. En effet sur le graphique de droite de la figure 8, le retard (r dans la formule 3) a été passé entre 8 et 12 alors qu'il était de 5 dans les simulations du graphique de gauche. Cette modification a grandement amélioré cette version, en lui permettant d'atteindre une récompense intrinsèque plus importante que la version uniforme. Cependant, il est important de souligner que la version uniforme de DIAYN a un temps d'apprentissage près de deux fois plus court que toutes nos versions. Nous avons également observé que la diversité des nouvelles compétences pouvait-être amélioré. Si les compétences apprises sont distinctes, elles restent tout de même proche du point de départ de l'agent. Nous avons donc cherché une méthode pour favoriser l'exploration.

5.5 Ajout d'un bonus d'exploration

En observant que ces variantes contribuaient peu à la diversification des compétences, nous avons pensé à ajouter un bonus d'exploration à la récompense. Un bonus d'exploration est une récompense qui est calculée à partir du nouvel état s' et qui est ajouté à la récompense intrinsèque de l'agent. L'objectif est de ne pas utiliser des connaissances dont l'agent n'a pas accès pour le calculer, afin que la récompense reste intrinsèque (ce qui permet à la méthode de rester applicable dans des cas réels). Le bonus d'explorations que nous avons ajouté est :

$$B_{exp}(s_{t+1}) = -log(p(s_{t+1})) \tag{4}$$

avec $p(s_{t+1})$ la probabilité de trouver le nouvel état s_{t+1} dans la mémoire épisodique.

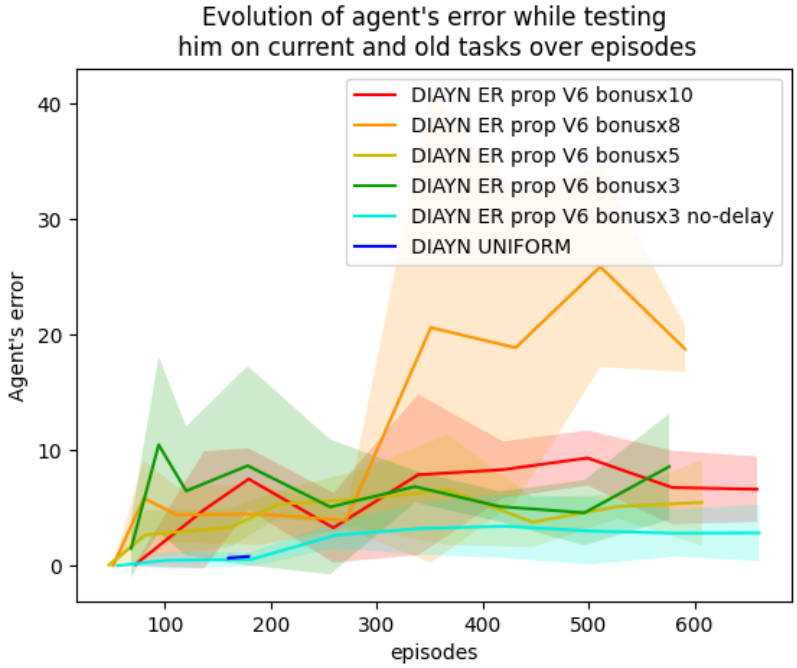

FIGURE 9: Évolution de l'erreur moyenne du discriminant de différents agents lorsqu'ils sont testés sur d'anciennes compétences.

Étant donné que le bonus d'exploration modifie la récompense intrinsèque de l'agent, nous ne pouvions plus observer l'évolution de cette variable pour évaluer notre méthode. Nous avons donc choisie d'observer l'erreur du discriminant, qui évolue de la même manière sans pour autant être biaisée par le bonus d'exploration. Nous avons donc testé cette méthode, en modifiant le rapport r_{bon} entre la récompense intrinsèque et le bonus d'exploration pour obtenir une nouvelle récompense intrinsèque correspondant à :

$$R'_{int} = R_{int}(s_{t+1}, c) + r_{bon} * B_{exp}(s_{t+1}) \tag{5}$$

avec R'_{int} la nouvelle récompense intrinsèque de l'agent, R_{int} l'ancienne récompense intrinsèque de l'agent, et B_{exp} la valeur du bonus d'exploration. Cette variante a été comparé à la version uniforme également (Cf. figure 9 "X n" dans le nom de l'agent indique r_{bon} = n).

Nous pouvons ici remarquer que l'erreur est équivalente à celle de DIAYN, sans être meilleure. Cependant, l'ajout du bonus d'exploration a permis d'obtenir des compétences plus éloignés du point de départ de l'agent, même si ces compétences sont moins distinctes (cf

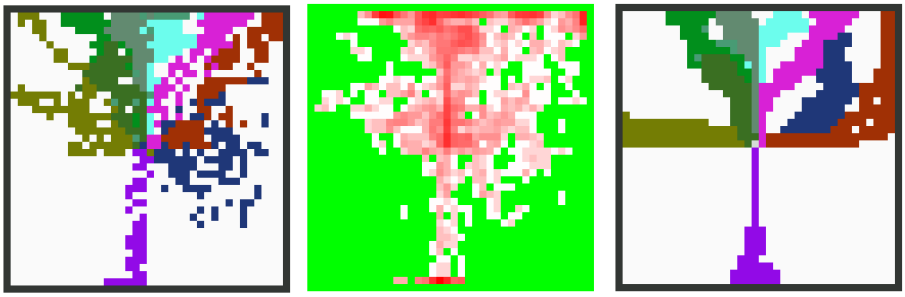

FIGURE 10: De gauche à droite, représentation des données se trouvant dans la mémoire épisodique de l'agent à la fin de l'apprentissage, la représentation de la répartition du bonus d'exploration (du rouge pour un bonus faible au vert pour un bonus élevé), puis l'exécution des compétences apprises à la fin de l'apprentissage (5 exécution par compétences toutes superposées).

En testant ces méthodes, nous nous sommes aperçu qu'elles permettaient d'égaler la version uniforme de DIAYN en observant l'erreur du discriminant, mais mettait tout de même plus de temps à apprendre.

Les variantes que nous avons proposés présentes des avantages et des inconvénients l'une par rapport à l'autre. Il sera nécessaire pour la suite du stage de trouver les paramètres optimaux pour chacune d'elles, et de combiner leurs avantages en corrigeant leur défauts.

5.6 Tests de combined replay appliqué à DIAYN

L'idée d'application de combined replay à DIAYN 4.3 n'a pas encore été implémentée. Une collaboration est en train d'être initialisée avec les auteurs de l'article *combined replay* [29] (CEA - Grenoble) pour utiliser leur savoir faire dans l'implémentation de cette méthode.

6 Conclusion

Pour prendre du recul sur les contributions et le travail réalisé, il est important de souligner que les applications de *experience replay* à DIAYN ont permis d'améliorer la motivation intrinsèque de l'agent et donc la discriminabilité des compétences apprises, par rapport à la version uniforme, tout en observant très peu d'oubli. Au sein de l'état de l'art des méthodes corrigeant l'oubli, seul P.N.N. [27] obtenait un tel résultat, au prix d'un réseaux de neurones par classe et d'un complexité en espace très importante. De plus, notre méthode fonctionne en CIL ce qui n'était pas le cas de PNN.

Cependant, notre méthode conserve d'important défauts par rapport à la version uniforme. Premièrement, le temps d'apprentissage est près de deux fois

plus long pour notre méthode. Ensuite, les compétences apprises peinent à être aussi exploitable que les compétences apprises de manière uniforme. En effet, nos compétences sont autant voir plus (selon les implémentations) discriminable que celles de la version uniforme, mais ne semble pas, selon nos observation, recouvrir aussi bien l'espace d'états. Une étude plus poussée avec de nouvelle mesures pourrais permettre de corriger ces défauts.

Enfin, une marge d'amélioration très importante est espéré avec l'implémentation de combined replay, qui pourrais permettre d'apprendre moins de compétences pour un résultat équivalent, et d'apprendre sur des données plus générales et réduisant le sur-apprentissage, tout en en conservant moins en mémoire.

Remerciements

Ce travail a été financé par la chaire IA "Remember" (ANR-20-CHIA-0018) de l'ANR.

Références

1. Achiam, J., Edwards, H., Amodei, D., Abbeel, P. : Variational option discovery algorithms. arXiv preprint arXiv :1807.10299 (2018)
2. Ans, B., Rousset, S. : Avoiding catastrophic forgetting by coupling two reverberating neural networks. Comptes Rendus de l'Académie des Sciences-Series III-Sciences de la Vie 320(12), 989–997 (1997)
3. Aubret, A., Matignon, L., Hassas, S. : A survey on intrinsic motivation in reinforcement learning. arXiv preprint arXiv :1908.06976 (2019)
4. Aubret, A., Matignon, L., Hassas, S. : Elsim : End-to-end learning of reusable skills through intrinsic motivation. arXiv preprint arXiv :2006.12903 (2020)
5. Campos, V., Trott, A., Xiong, C., Socher, R., Giro-i Nieto, X., Torres, J. : Explore, discover and learn : Unsupervised discovery of state-covering skills. In : International Conference on Machine Learning. pp. 1317–1327. PMLR (2020)
6. Chaudhry, A., Ranzato, M., Rohrbach, M., Elhoseiny, M. : Efficient lifelong learning with a-gem. arXiv preprint arXiv :1812.00420 (2018)
7. Chaudhry, A., Rohrbach, M., Elhoseiny, M., Ajanthan, T., Dokania, P.K., Torr, P.H., Ranzato, M. : On tiny episodic memories in continual learning. arXiv preprint arXiv :1902.10486 (2019)
8. Colas, C., Fournier, P., Chetouani, M., Sigaud, O., Oudeyer, P.Y. : Curious : intrinsically motivated modular multi-goal reinforcement learning. In : International conference on machine learning. pp. 1331–1340. PMLR (2019)
9. Eysenbach, B., Gupta, A., Ibarz, J., Levine, S. : Diversity is all you need : Learning skills without a reward function. arXiv preprint arXiv :1802.06070 (2018)
10. Fernando, C., Banarse, D., Blundell, C., Zwols, Y., Ha, D., Rusu, A.A., Pritzel, A., Wierstra, D. : Pathnet : Evolution channels gradient descent in super neural networks. arXiv preprint arXiv :1701.08734 (2017)
11. Fujimoto, S., Hoof, H., Meger, D. : Addressing function approximation error in actor-critic methods. In : International Conference on Machine Learning. pp. 1587–1596. PMLR (2018)

12. Gregor, K., Rezende, D.J., Wierstra, D. : Variational intrinsic control. arXiv preprint arXiv :1611.07507 (2016)
13. Grossberg, S. : How does a brain build a cognitive code ? Studies of mind and brain pp. 1–52 (1982)
14. Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., et al. : Soft actor-critic algorithms and applications. arXiv preprint arXiv :1812.05905 (2018)
15. Hochreiter, S., Schmidhuber, J. : Long short-term memory. Neural computation 9(8), 1735–1780 (1997)
16. Kearns, M., Singh, S. : Near-optimal reinforcement learning in polynomial time. Machine learning 49(2), 209–232 (2002)
17. Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A.A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., et al. : Overcoming catastrophic forgetting in neural networks. Proceedings of the national academy of sciences 114(13), 3521–3526 (2017)
18. Kutalev, A. : Natural way to overcome the catastrophic forgetting in neural networks. arXiv preprint arXiv :2005.07107 (2020)
19. Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D. : Continuous control with deep reinforcement learning. arXiv preprint arXiv :1509.02971 (2015)
20. Liu, G.K. : Weight friction : A simple method to overcome catastrophic forgetting and enable continual learning. arXiv preprint arXiv :1908.01052 (2019)
21. Mallya, A., Lazebnik, S. : Packnet : Adding multiple tasks to a single network by iterative pruning. In : Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 7765–7773 (2018)
22. Masana, M., Liu, X., Twardowski, B., Menta, M., Bagdanov, A.D., van de Weijer, J. : Class-incremental learning : survey and performance evaluation. arXiv preprint arXiv :2010.15277 (2020)
23. McCloskey, M., Cohen, N.J. : Catastrophic interference in connectionist networks : The sequential learning problem. In : Psychology of learning and motivation, vol. 24, pp. 109–165. Elsevier (1989)
24. Mnih, V., Badia, A.P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., Kavukcuoglu, K. : Asynchronous methods for deep reinforcement learning. In : International conference on machine learning. pp. 1928–1937. PMLR (2016)
25. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., et al. : Human-level control through deep reinforcement learning. nature 518(7540), 529–533 (2015)
26. Nair, A., Pong, V., Dalal, M., Bahl, S., Lin, S., Levine, S. : Visual reinforcement learning with imagined goals. arXiv preprint arXiv :1807.04742 (2018)
27. Rusu, A.A., Rabinowitz, N.C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., Hadsell, R. : Progressive neural networks. arXiv preprint arXiv :1606.04671 (2016)
28. Ryan, R.M., Deci, E.L. : Intrinsic and extrinsic motivations : Classic definitions and new directions. Contemporary educational psychology 25(1), 54–67 (2000)
29. Solinas, M., Rousset, S., Cohendet, R., Bourrier, Y., Mainsant, M., Molnos, A., Reyboz, M., Mermillod, M. : Beneficial effect of combined replay for continual learning. In : ICAART (2). pp. 205–217 (2021)
30. Tsitsiklis, J.N. : Asynchronous stochastic approximation and q-learning. Machine learning 16(3), 185–202 (1994)
31. Vitter, J.S. : Random sampling with a reservoir. ACM Transactions on Mathematical Software (TOMS) 11(1), 37–57 (1985)

7 Annexe 1 - Information mutuelle

Inspiré de la revue [3]. L'information mutuelle entre deux variable aléatoire X et Y est noté $I(X,Y)$. Elle permet de quantifier l'information contenu dans une variable aléatoire X au sujet d'une autre variable aléatoire Y. Elle peut-être vu comme la décroissance de désordre apporté par une variable aléatoire Y sur une variable aléatoire X.

$$I(X,Y) = H(X) - H(X|Y) \tag{6}$$

Avec H l'entropie de Shannon quantifiant l'information nécessaire moyenne pour déterminer la valeur d'une variable aléatoire. Soit X, une variable aléatoire avec pour loi de densité $p(x)$, son entropie est alors définie par :

$$H(X) = -\int_x p(x)log(p(x)) \tag{7}$$

8 Annexe 2 - Paramètres des simulations

Le *seuil d'erreur* est une valeur permettant à un agent, lors d'un apprentissage séquentiel, de passer à la séquence suivante lorsque l'erreur moyenne du discriminant lors des 20 derniers interactions passe sous ce seuil. SAC 2.1 correspond à l'algorithme d'apprentissage par renforcement utilisé. Sa température indique à quel point l'entropie du réseau acteur est prise en compte dans l'évaluation d'un état. *tau* correspond à la vitesse avec laquelle le réseau target tend vers le réseau principale dans l'algorithme SAC.

8.1 Experience replay - première simulation

Simulation présentée en figure 5.

8.2 Experience replay - seconde simulation

Simulation présentée en figure 6.

8.3 Experience replay - troisième simulation

Simulation présentée en figure 8.

8.4 Experience replay - Quatrième simulation

Simulation présentée en figure 9 et 10.

paramètre	valeur
Paramètres généraux de la simulation	
nombre de tests par agent	5
seuil d'erreur	0.2
nombre de compétences	10
nombre maximum d'épisodes par compétences	70
nombre de compétences dans la première séquence	2
nombre de compétences dans les séquences suivantes	1
Paramètres communs à toutes les implémentations	
taux d'apprentissage (tout réseaux)	0.003
taille de la première couche (tout réseaux)	128
taille de la seconde couche (tout réseaux)	64
taille du buffer du discriminant (en données maximales)	1000
Taille des batch d'apprentissage (RL et Discriminant	128)
gamma	0.99
tau	0.005
Paramètres propres aux implémentations sans experience replay	
Température de SAC	1
taille du buffer de SAC (en données maximales)	10000
Paramètres propres aux implémentations utilisant experience replay	
Température de SAC	1.5
taille du buffer de SAC (en données maximales)	2000
taille de la mémoire épisodique	500

TABLE 6: Paramètres de la première simulation de experience replay

paramètre	valeur
Paramètres généraux de la simulation	
seuil d'erreur	0.2
nombre de compétences	10
nombre maximum d'épisodes par compétences	70
nombre de compétences dans la première séquence	2
nombre de compétences dans les séquences suivantes	1
Paramètres de l'implémentation	
taux d'apprentissage (tout réseaux)	0.003
taille de la première couche (tout réseaux)	128
taille de la seconde couche (tout réseaux)	64
taille du buffer du discriminant (en données maximales)	1000
Taille des batch d'apprentissage (RL et Discriminant)	128
Température de SAC	1
gamma	0.99
tau	0.005
taille du buffer de SAC (en données maximales)	2000
taille de la mémoire épisodique	2000

TABLE 7: Paramètres de la seconde simulation de experience replay

paramètre	valeur
Paramètres généraux de la simulation	
nombre de tests par agent	5
seuil d'erreur	0.15
nombre de compétences	10
nombre maximum d'épisodes par compétences	70
nombre de compétences dans la première séquence	2
nombre de compétences dans les séquences suivantes	1
Paramètres communs à toutes les implémentations	
taux d'apprentissage (tout réseaux)	0.003
taille de la première couche (tout réseaux)	128
taille de la seconde couche (tout réseaux)	64
taille du buffer du discriminant (en données maximales)	1000
Taille des batch d'apprentissage (RL et Discriminant	128)
gamma	0.99
tau	0.005
Paramètres propres aux implémentations sans experience replay	
Température de SAC	1
taille du buffer de SAC (en données maximales)	10000
Paramètres propres aux implémentations utilisant experience replay	
Température de SAC	1.5
taille du buffer de SAC (en données maximales)	2000
taille de la mémoire épisodique	2000
retard du délais	5 (ou selon le nom de l'agent testé)
vitesse du délais	.5

TABLE 8: Paramètres de la troisième simulation de experience replay

paramètre	valeur
Paramètres généraux de la simulation	
nombre de tests par agent	5
seuil d'erreur	0.15
nombre de compétences	10
nombre maximum d'épisodes par compétences	70
nombre de compétences dans la première séquence	2
nombre de compétences dans les séquences suivantes	1
Paramètres communs à toutes les implémentations	
taux d'apprentissage (tout réseaux)	0.003
taille de la première couche (tout réseaux)	128
taille de la seconde couche (tout réseaux)	64
taille du buffer du discriminant (en données maximales)	1000
Taille des batch d'apprentissage (RL et Discriminant	128)
gamma	0.99
tau	0.005
Paramètres propres aux implémentations sans experience replay	
Température de SAC	1
taille du buffer de SAC (en données maximales)	10000
Paramètres propres aux implémentations utilisant experience replay	
Température de SAC	1.5
taille du buffer de SAC (en données maximales)	2000
taille de la mémoire épisodique	2000
retard du délais	7
vitesse du délais	.4

TABLE 9: Paramètres de la quatrième simulation de experience replay