Reinforcement Learning & Advanced Deep

M2 DAC

2020-2021

TME 12 : Implicit Curriculum RL

De nombreuses tâches en RL mettent en jeu des environnements à récompenses parcimonieuses, c'est à dire délivrant des récompenses non nulles uniquement dans les rares cas où l'objectif visé est parfaitement accompli par l'agent. Ce genre d'environnement est particulièrement difficile à traiter par des approches de renforcement classiques, car l'agent n'obtient aucun signal lui permettant de donner une direction à sa progression tant qu'il n'atteint pas le but visé. Or ce but visé peut être complexe à atteindre et des explorations classiques de type epsilon-greedy peuvent ne pas permettre l'obtention de la moindre récompense tout au long du processus d'apprentissage.

Pour s'en rendre compte on se propose de considérer des problèmes GridWorld dirigés par des buts. Plutôt que de travailler avec l'environnement GridWorld donnant des récompenses de manière fixe selon les états rencontrés par l'agent, l'idée est d'apprendre une politique universelle $\pi(a_t|s_t, g)$ qui vise à atteindre un état but g échantillonné de l'environnement en début de chaque nouvel épisode via :

```
goal, _= env.sampleGoal()
goal=featureExtractor.getFeatures(goal)
```

Cette fonction *env.sampleGoal()* échantillonne uniformément un but selon une liste de buts chargée en même tant que la carte de l'environnement. Utiliser pour cela les fichiers de carte du répertoire *gridworldPlans&Goals* fourni avec l'environnement sur le site de l'UE. Ces fichiers contiennent d'abord la carte de départ de l'agent, puis une liste de situations but que l'on souhaite pouvoir atteindre à la demande.

Une fois ces cartes chargées, on travaillera en ignorant les retours *reward* et *done* de la fonction step de l'environnement, et en les remplaçant par un test d'atteinte du but :

```
action = agent.act(ob, goal)
new_ob, _, _, _= env.step(action)
new_ob = agent.featureExtractor.getFeatures(new_ob)
done=(new_ob==goal).all()
reward=1.0 if done else -0.1
```

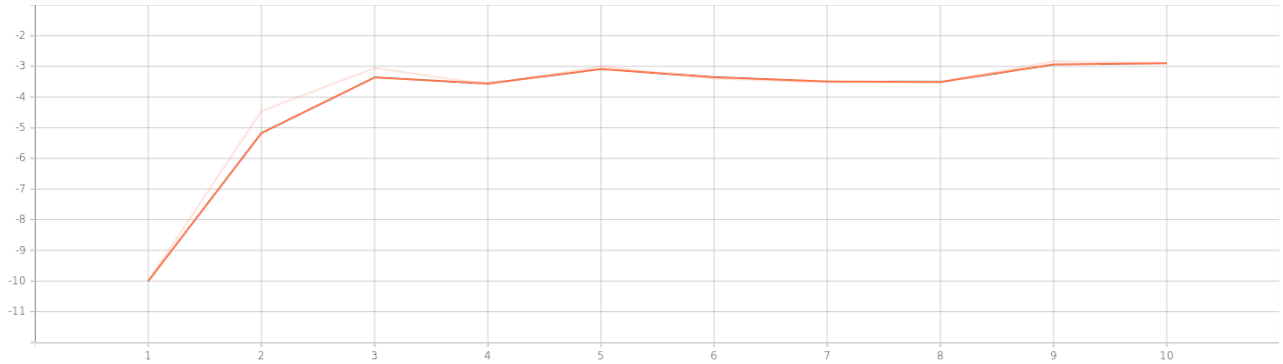
1 DQN avec buts

On se propose ici de mettre en œuvre une version de DQN considérant des buts (Universal Value Function Approximators [1]). Il s'agit alors de considérer une fonction $Q : \mathcal{A} \times \mathcal{S} \times \mathcal{G} \to \mathbb{R}$, où $\mathcal{A}$ est l'ensemble des actions, $\mathcal{S}$ l'ensemble des états et $\mathcal{G}$ l'ensemble des buts à considérer. Une manière d'étendre simplement le code de DQN à ce cadre est de se ramener à $Q_\phi : \mathcal{A} \times \mathcal{S}' \to \mathbb{R}$ en considérant $\mathcal{S}'$ un ensemble d'états-but dont chaque élément correspond à la concaténation d'un état de $\mathcal{S}$ avec un but de $\mathcal{G}$. Le code du DQN classique implémenté en semaine 3 peut alors être réutilisé tel quel (avec une taille d'état d'entrée du réseau doublée).

Lorsque les buts sont relativement bien répartis sur l'ensemble de l'espace, l'apprentissage d'un agent efficace se passe relativement bien, si tant est qu'il puisse exister une relation de corrélation entre distance entre états $d_{\mathcal{S}}(x, y)$ (en terme de nombre minimal d'actions à effectuer pour passer de l'état x à l'état y dans l'environnement) et distances euclidiennes $||\phi(x) - \phi(y)||_2$ entre représentations ϕ produites par les réseaux de neurones considérés. Cette propriété permet la généralisation des connaissances sur les valeurs d'actions entre les différents buts de l'environnement. Cependant, à moins de considérer des couches de neurones convolutionnelles (CNN) en entrée du réseau Q_ϕ (permettant d'assurer une notion d'équivariance dans les représentations intermédiaires manipulées), ce n'est pas le cas lorsque l'on utilise la sortie de le fonction $utils.MapFromDumpExtractor2$ en entrée du réseau (la distance euclidienne entre deux représentations est la même quel que soit le déplacement

de l'agent entre les deux états considérés, que ce soit d'une seule case ou bien de $k > 1$ cases). On propose alors de considérer une représentation plus efficace donnée par $utils.MapFromDumpExtractor4$, qui retourne simplement les coordonnées x et y de l'agent dans l'état considéré. C'est cette représentation des états et des buts, par ailleurs bien plus légère, qui est utilisée dans la suite.

On se propose de considérer dans un premier temps le fichier de carte *plan2Multi.txt*. Ce fichier possède des buts proches de l'agent de départ, et tout au long du chemin menant aux buts les plus éloignés, ce qui permet d'avoir des récompenses pour progresser. L'apprentissage de DQN pour ce problème devrait vous donner une courbe de résultats similaire à la suivante, dont chaque point correspond à la moyenne des récompenses obtenues sur 100 épisodes de test tous les milliers d'épisodes d'apprentissage (le point 1 correspond à la moyenne avant le début de l'apprentissage, le point 5 correspond à la moyenne après 4000 épisodes d'apprentissage) :

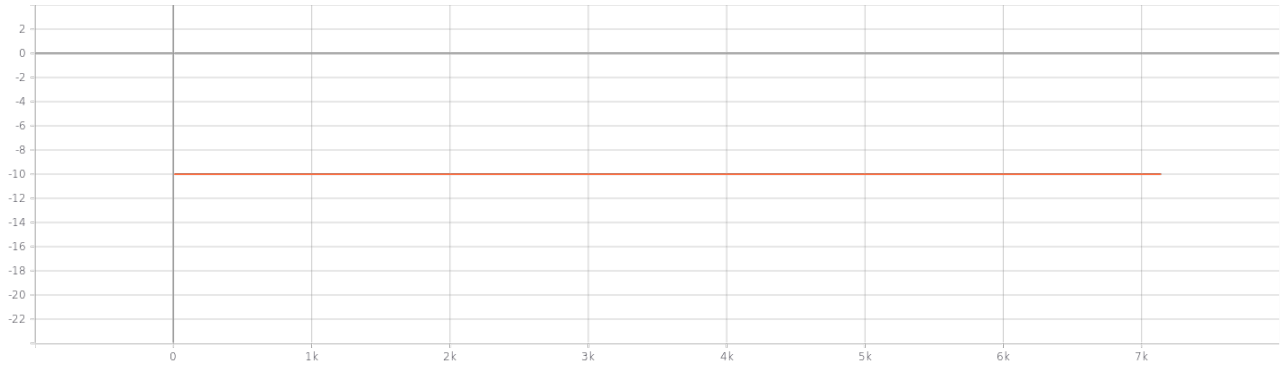


Cette courbe, ainsi que les suivantes sauf spécifié autrement, a été obtenue sur des épisodes de taille maximale 100 avec un doubleDQN utilisant un réseau à deux couches cachées de 200 neurones chacune et activations tanh, un pas d'apprentissage de 1E-3, un facteur d'exploration de 0.2 (sans decay), un replay buffer de 1E6 transitions, un facteur de discount $\gamma = 0.99$, des mini-batchs de taille 1000, une mise à jour du réseau cible tous les 1000 évènements et un pas de gradient effectué tous les 10 évènements (différence temporelle avec torch.nn.MSELoss), sans Prioritized Replay.

On observe un bon apprentissage qui mène à un agent quasi-parfait pour atteindre chacun des buts, y compris le plus éloigné.

2 Hinsight Experience Replay

Les choses se compliquent lorsque l'environnement ne fournit pas de buts tout au long du chemin pour découvrir les objectifs les plus éloignés. On considère maintenant le fichier *plan2.txt*, qui charge la même carte que *plan2Multi.txt* mais avec un seul but visé, celui le plus difficile à atteindre pour l'agent (en case [33,38]). Dans ce cas, le même apprentissage qu'à la section précédente donne une progression nulle :



Contrairement à la configuration de *plan2Multi.txt*, il n'y a ici plus d'objectifs intermédiaires permettant de guider l'apprentissage vers les buts plus complexes à atteindre. Dans *plan2Multi.txt*, le fait de réussir à accomplir un but intermédiaire permet 1) de retourner explorer dans cette zone plus facilement par la suite et ainsi avoir plus de chances de découvrir le but suivant et 2) de donner une indication sur les moyens d'atteindre des buts non découverts par leur proximité avec les objectifs atteints. Ici en l'absence de buts intermédiaires, et malgré un fort coefficient d'exploration, l'agent est incapable d'atteindre le but, n'obtenant jamais aucun retour positif de ses expériences.

Hinsight Experience Replay (HER) [2] fournit un mécanisme permettant de dépasser ces limites dans divers cas. L'idée de HER est simple, il s'agit d'ajouter dans le Replay Buffer, en plus des transitions de chaque épisode

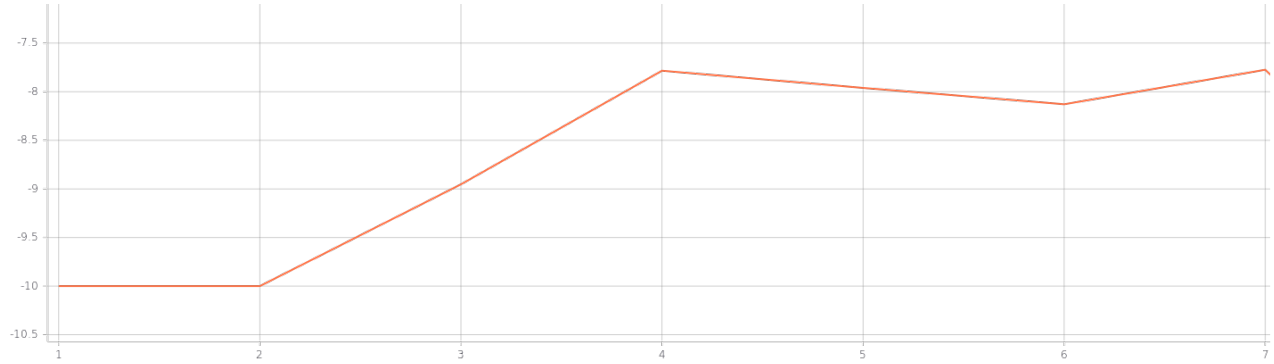
associées au but de l'environnement visé par l'épisode, les mêmes transitions mais cette fois associées à un état atteint dans la trajectoire de l'agent (généralement le dernier état rencontré par l'agent dans l'épisode). Cet état but ayant été atteint dans l'épisode (puisqu'il en fait partie), cela permet d'effectuer les mises à jour du réseau Q_ϕ avec des retours non nuls, la transition de l'épisode vers cet état étant alors associée à une récompense positive (+1). L'idée est d'exploiter les connaissances sur des sous-buts déjà atteints pour généraliser aux buts réels de l'environnement non encore rencontrés.

Algorithm 1 Hindsight Experience Replay (HER)

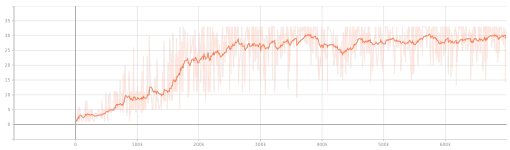
Given:
- an off-policy RL algorithm $\mathbb{A}$, ▷ e.g. DQN, DDPG, NAF, SDQN
- a strategy $\mathbb{S}$ for sampling goals for replay, ▷ e.g. $\mathbb{S}(s_0, \ldots, s_T) = m(s_T)$
- a reward function $r : \mathcal{S} \times \mathcal{A} \times \mathcal{G} \to \mathbb{R}$. ▷ e.g. $r(s, a, g) = -[f_g(s) = 0]$

Initialize $\mathbb{A}$ ▷ e.g. initialize neural networks
Initialize replay buffer R
for episode = 1, M **do**
  Sample a goal g and an initial state s_0.
  for $t = 0, T-1$ **do**
    Sample an action a_t using the behavioral policy from $\mathbb{A}$:
      $a_t \leftarrow \pi_b(s_t||g)$ ▷ $||$ denotes concatenation
    Execute the action a_t and observe a new state s_{t+1}
  end for
  for $t = 0, T-1$ **do**
    $r_t := r(s_t, a_t, g)$
    Store the transition $(s_t||g,\ a_t,\ r_t,\ s_{t+1}||g)$ in R ▷ standard experience replay
    Sample a set of additional goals for replay $G := \mathbb{S}$(**current episode**)
    for $g' \in G$ **do**
      $r' := r(s_t, a_t, g')$
      Store the transition $(s_t||g',\ a_t,\ r',\ s_{t+1}||g')$ in R ▷ HER
    end for
  end for
  for $t = 1, N$ **do**
    Sample a minibatch B from the replay buffer R
    Perform one step of optimization using $\mathbb{A}$ and minibatch B
  end for
end for

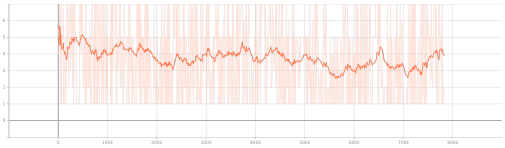
La mise en oeuvre de l'algorithme HER donné ci-dessus avec DQN (qui diffère de DQN uniquement au niveau de la boucle sur des buts échantillonnés de l'épisode courant), avec G ne contenant que l'état final de chaque épisode, une récompense associée de +1 lorsque la transition mène à cet état de G et toutes les autres récompenses à -0.1, permet d'obtenir la courbe de résultats de test suivante (tous les autres paramètres identiques à ceux décrits en section précédente), ce qui correspond finalement à un agent quasi parfait pour l'accomplissement du but de l'environnement :

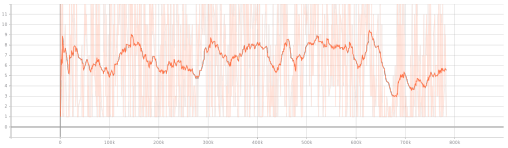


Si on trace l'évolution des coordonnées (x,y) du dernier état atteint dans chaque épisode lors de cet apprentissage, on note par ailleurs une claire évolution vers le point visé [33,38] (coordonnées x à gauche, y à droite) :



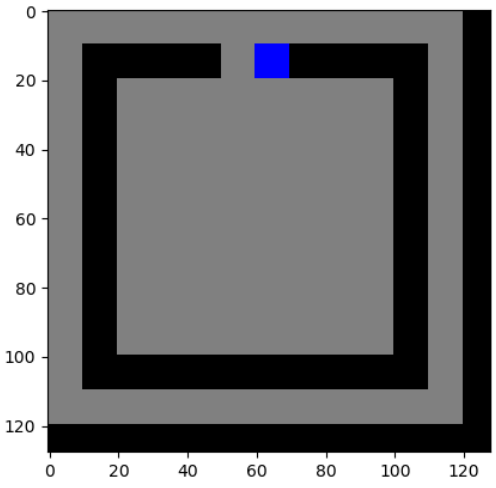
Alors que sans HER, on stagne proche de la zone de départ (en [1,1]) :



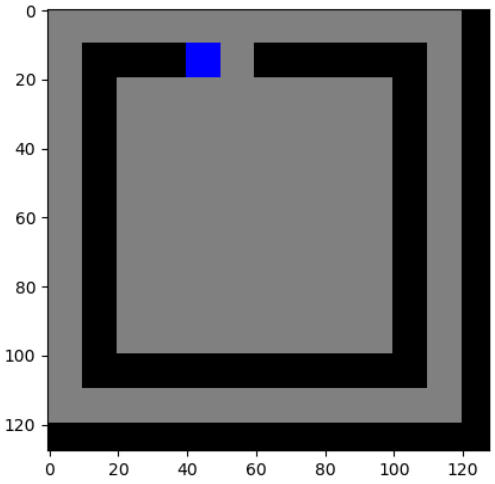


3 Échantillonnage itératif de buts

Un exemple de cas difficile pour HER est donné dans le fichier *plan3.txt*, qui charge la carte et le but suivants :



Situation de départ



État objectif

Dans ce cas, il faut dans un premier temps accepter de s'éloigner du but (en terme de distance euclidienne des représentations des états), pour pouvoir l'atteindre. Or, HER tend à faire apprendre au processus que d'aller à droite à partir de la situation de départ permet d'atteindre des états plus éloignés du but visé (selon toujours une représentation x,y de l'agent). Le processus tend alors à converger vers la situation de départ de manière constante sur les épisodes, l'agent choisissant quasi toujours de se déplacer vers la gauche (contre le mur donc), empêchant toute progression.

Une des limites de HER paraît être le fait que l'échantillonnage des trajectoires se fait toujours en fonction des buts réels de l'environnement. Or, dans des situations où les buts atteints au cours du processus ne permettent pas de bien inférer des directions à prendre pour atteindre les zones cible, le processus s'enferme dans des optima locaux. Une proposition pour dépasser cela serait d'échantillonner des trajectoires, en fonction de buts fictifs, adaptés au niveau de l'agent.

Pour chaque épisode, avec une probabilité β, on propose donc d'échantillonner de manière itérative des buts à atteindre à partir d'un *buffer* de buts déjà rencontrés, en priorisant les états candidats en fonction de leur niveau de difficulté : on ne souhaite ni échantillonner des buts trop simples, auquel cas on n'en apprendrait pas grand chose de nouveau, ni trop compliqués, car ne menant à aucun signal de progression possible. Ainsi, avec n_i le nombre de tentatives déjà effectuées pour le but candidat i et v_i le nombre de succès parmi ces tentatives, on considère la probabilité de sélectionner ce but dans le buffer selon $p_i \propto \exp(\alpha H_i)$ avec H_i l'entropie de la capacité de l'agent à atteindre cet état i :

$$H_i = -\frac{v_i}{n_i} \times \log(\frac{v_i}{n_i}) - (1 - \frac{v_i}{n_i}) \times \log(1 - \frac{v_i}{n_i})$$

où l'hyper-paramètre α correspond à un coefficient de température permettant de donner plus ou moins d'importance aux états d'entropie élevée dans la distribution d'échantillonnage (si $\alpha = 0$: distribution uniforme, si $\alpha \to \infty$: Dirac centré sur $\arg\max H_i$).

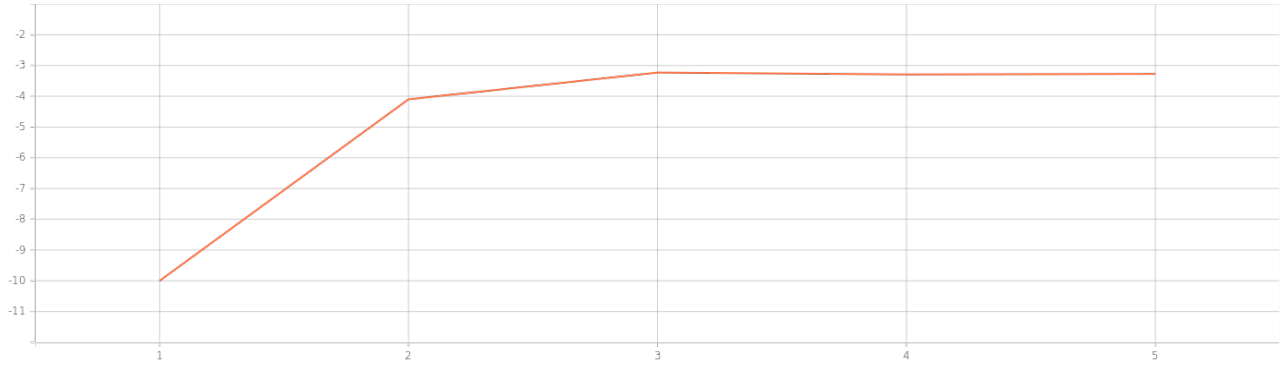
Ainsi, en plus du mécanisme de HER, l'idée est de ne pas nécessairement faire agir l'agent pour les buts réels (qui peuvent être complexes), mais le faire interagir avec l'environnement selon une politique dirigée vers un but adapté à son niveau. D'autre part, plutôt que de faire repartir l'agent de son point de départ après chaque but atteint, on échantillonne un nouveau but pour la suite de l'épisode (on repart de la position de départ uniquement lorsque le nombre d'évènements maximal pour un épisode est atteint). Cela permet de favoriser l'exploration dans des zones peu connues de l'agent, mais pas impossibles à atteindre pour lui, ce qui lui permet

de progresser vers les zones cibles de l'environnement. Bien sûr, pour rester connecté aux vrais objectifs de l'environnement, les buts réels restent régulièrement considérés dans l'algorithme, selon une probabilité $1 - \beta$ d'échantillonner un but réel plutôt qu'à partir du buffer de buts. L'algorithme ci-dessous détaille le processus. On note que pour chaque but g visé au cours d'un épisode, toutes les transitions depuis le début de l'épisode jusqu'à l'instant où g est atteint (ou que le nombre d'actions maximal est dépassé) sont ajoutées dans le buffer associées à g (i.e., τ n'est vidé qu'en début d'épisode dans l'algorithme ci-dessous). Cela permet de considérer toute la trajectoire depuis s_0 pour chacun des buts, et donc ainsi acquérir de l'expérience sur des trajectoires entières, y compris pour des buts difficiles à atteindre directement, même si le choix des actions se faisait selon des buts différents sur le début de l'épisode.

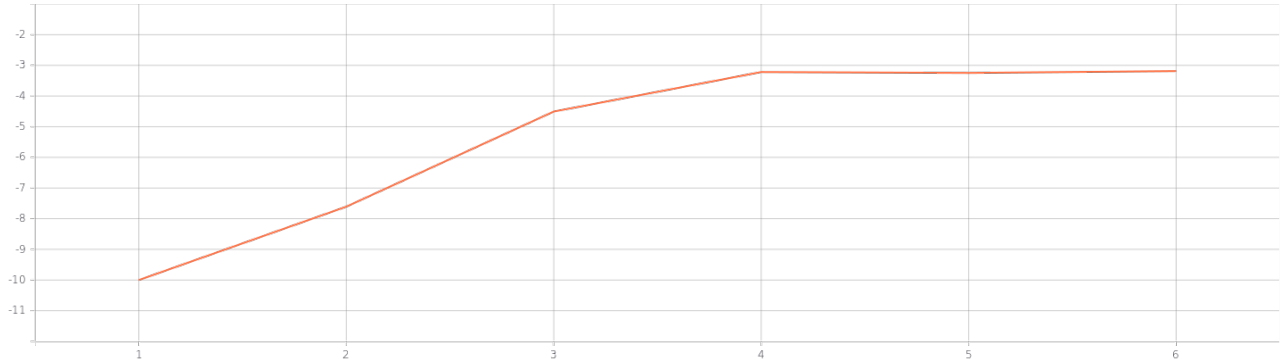
Algorithm 1: Iterative Goal Sampling

$\mathcal{R} \leftarrow \emptyset$, $\mathcal{G} \leftarrow \emptyset$
for *i=0,1,2,...* **do**
  $g \leftarrow \emptyset$, $t \leftarrow 0$
  $\tau \leftarrow \emptyset$ `// Buffer of transitions for the current episode`
  Sample s_0 from the environnement
  while $t < maxT$ **do**
    if $g = \emptyset$ **then** `// Goal Sampling`
      $b \sim Bernouilli(\beta)$
      if *1-b* **then**
        Sample goal g from the environnement
      end
      else
        Sample goal g from buffer $\mathcal{G}$, with $p_g \propto exp(\alpha H_g)$
      end
    end
    $a_t \sim \pi(a_t|s_t, g)$
    Execute a_t and observe new state s_{t+1}
    Store (s_t, a_t, s_{t+1}) in τ
    if *g reached at* s_{t+1} *or* $t \geq (maxT - 1)$ **then** `// Store Experience for g`
      for $(s, a, s') \in \tau$ **do**
        $r = 1$ if g reached at s', $r = -0.1$ else
        Store $(s||g, a, r, s'||g)$ in $\mathcal{R}$
      end
      Optionally store transitions from τ in $\mathcal{R}$ with additional goals as in HER
      if b **then** `// Update priority in` $\mathcal{G}$
        $n_g \leftarrow n_g + 1$
        if *g reached at* s_{t+1} **then**
          $v_g \leftarrow v_g + 1$
        end
      end
      else break;
      $g \leftarrow \emptyset$
    end
    $t \leftarrow t + 1$
  end
  if *It's time to feed* $\mathcal{G}$ **then** `// Feed` $\mathcal{G}$
    if $s_t \notin \mathcal{G}$ **then** Store s_t in $\mathcal{G}$;
    else Wait next episode to add a new goal in $\mathcal{G}$;
  end
  if *It's time to learn* **then** `// Learn` π
    Optimize π w.r.t. experience in $\mathcal{R}$
  end
end

Cet échantillonnage itératif de buts permet d'obtenir la courbe de résultats en test suivante (uniquement sur buts réels) dans la configuration du fichier *plan3.txt*, avec $\alpha = 3$, $\beta = 0.9$ et un buffer de 10 buts fictifs (liste FIFO) que l'on alimente tous les 10 épisodes avec le dernier état atteint, là où HER seul échoue à obtenir le moindre reward positif :



À noter que HER (le fait d'ajouter des transitions associée à l'état atteint comme but fictif dans le Replay Buffer de transitions, ligne 24 de l'algorithme), bien que permettant une convergence un peu plus rapide, n'est pas indispensable lorsque l'on échantillonne des buts fictifs à atteindre pour l'agent. Voici la courbe de test pour la version avec échantillonnage de but décrit dans cette section, mais sans le mécanisme de HER :



4 Bonus : GoalGAN

Une autre possibilité pour ce genre de cas est d'employer l'approche GoalGAN [3], donnée dans l'algorithme ci-dessous :

Algorithm 1 Generative Goal Learning

Input: Policy π_0
Output: Policy π_N
$(G, D) \leftarrow$ `initialize_GAN()`
$goals_{\text{old}} \leftarrow \varnothing$
for $i \leftarrow 1$ **to** N **do**
  $z \leftarrow$ `sample_noise`$(p_z(\cdot))$
  $goals \leftarrow G(z) \cup$ `sample`$(goals_{\text{old}})$
  $\pi_i \leftarrow$ `update_policy`$(goals, \pi_{i-1})$
  $returns \leftarrow$ `evaluate_policy`$(goals, \pi_i)$
  $labels \leftarrow$ `label_goals`$(returns)$
  $(G, D) \leftarrow$ `train_GAN`$(goals, labels, G, D)$
  $goals_{\text{old}} \leftarrow$ `update_replay`$(goals)$
end for

L'idée, similaire à celle de la section précédente, est de chercher à échantillonner des buts adaptés au niveau de l'agent. Cependant, plutôt que de se limiter à des états déjà rencontrés comme buts candidats, il s'agit d'apprendre un générateur de buts, via un mécanisme de type GAN, où un discriminateur est appris à prédire si un but est d'intérêt où non pour le processus d'apprentissage :

$$\min_D V(D) = \mathbb{E}_{g \sim p_{\text{data}}(g)} \left[y_g (D(g) - 1)^2 + (1 - y_g) (D(g) + 1)^2 \right] + \mathbb{E}_{z \sim p_z(z)} \left[(D(G(z)) + 1)^2 \right]$$

$$\min_G V(G) = \mathbb{E}_{z \sim p_z(z)}[D(G(z))^2]$$

où p_{data} correspond à l'ensemble des buts du buffer de buts précédemment générés (initialisés avec des états rencontrés lors d'une phase d'exploration initiale) et y_g est égal à 1 si le ratio d'épisodes testés avec le but g est inclus dans $[R_{min}; R_{max}]$, 0 sinon. Ainsi, on vise à faire tendre les scores du discriminateur vers 1 pour les buts de difficulté moyenne pour la politique en cours et vers -1 pour les buts trop faciles ou trop difficiles à atteindre, ainsi que les buts générés par le générateur en cours. Le générateur vise à générer là où le discriminateur est indécis, c'est à dire dans les zones où il prédit 0. À chaque itération, des anciens buts sont ajoutés aux nouveaux issus du générateur (1/3 d'anciens pour 2/3 de nouveaux) pour éviter des problèmes de *catastrophic forgetting*.

L'intérêt de l'approche est d'être à même de découvrir des buts potentiellement intéressants sans même les avoir rencontrés auparavant. Il paraît cependant assez difficile à stabiliser.

Références

[1] T. Schaul, D. Horgan, K. Gregor, and D. Silver, "Universal value function approximators," in *International conference on machine learning*. PMLR, 2015, pp. 1312–1320.

[2] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, and W. Zaremba, "Hindsight experience replay," *CoRR*, vol. abs/1707.01495, 2017. [Online]. Available : http://arxiv.org/abs/1707.01495

[3] C. Florensa, D. Held, X. Geng, and P. Abbeel, "Automatic goal generation for reinforcement learning agents," in *International conference on machine learning*. PMLR, 2018, pp. 1515–1528.