
Rapport PANDROIDE M1

---

# Outils de visualisation pour l'étude des pas de gradient et pour les sous-espaces de l'espace de politiques en apprentissage par renforcement profond

---

***Étudiants :***
Andy Torres
Paul-Tiberiu IORDACHE

***Encadré par :***
Olivier SIGAUD

16 mai 2024

# Table des matières

# 1 Introduction

Ce projet s'attache à la création d'outils de visualisation sophistiqués dédiés à l'exploration des performances des politiques et de l'étude des pas de gradient dans le contexte de l'apprentissage par renforcement.

L'objectif central est de concevoir un outil de visualisation permettant d'observer et d'analyser de manière approfondie la qualité des politiques au sein de sous-espaces spécifiques soigneusement définis au sein de l'espace global des politiques. Ces sous-espaces, représentés par des politiques qui régissent le comportement d'un agent, seront sélectionnés de manière judicieuse afin de fournir une compréhension détaillée de la dynamique et de la robustesse des politiques dans des contextes spécifiques d'apprentissage.

L'outil de visualisation envisagé offrira ainsi une représentation visuelle puissante et informative, facilitant la prise de décision et l'optimisation des politiques dans le domaine de l'apprentissage par renforcement.

Nous voulons préciser que l'outil permettant de visualiser les histogrammes est adapté seulement pour l'algorithme DQN. En revanche, celui pour la visualisation des sous-espaces de l'espace des politiques est fonctionnel pour n'importe quel algorithme d'apprentissage par renforcement. Le fait d'avoir généralisé le code pour qu'il soit facilement adaptable aux autres algorithmes de la bibliothèque BBRL permet de ne pas devoir apporter beaucoup de modifications par rapport au code pour rendre les outils fonctionnels pour les autres algorithmes d'apprentissage. Nous abordons également dans la suite de ce rapport les modifications à apporter pour l'implémentation dans d'autres algorithmes.

# 2 Histogrammes

Nous avons réalisé plusieurs histogrammes de manière à mesurer l'évolution de plusieurs grandeurs.

Ces grandeurs sont la norme euclidienne du gradient (le gradient représentant la direction et le taux de variation d'une fonction par rapport aux paramètres du modèle), la distance euclidienne entre deux politiques successives ainsi que la fonction de perte (*loss*) qui représente la différence entre les valeurs prédites par le modèle et les valeurs réelles calculées.

Ensuite, nous avons vérifié une hypothèse selon laquelle la norme du gradient serait proportionnelle à la distance euclidienne entre deux politiques successives, telle que : `distance` = `k` × `norme_gradient` avec `k` = `learning_rate` à une constante $\epsilon$ près.

Pour parvenir à ce résultat, nous nous sommes servis de 3 fichiers python distincts :

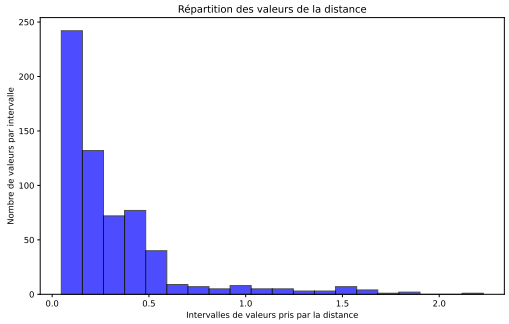
- `dqnv2.py` pour l'implémentation de l'algorithme DQN et le stockage des grandeurs dans des fichiers textes annexes distincts à l'aide de loggers. Chaque fichier texte (1 par grandeur) est ensuite stocké dans `/src/bbrl_algos/visualization/{nom_grandeur.txt}` → chaque ligne d'un document représente la valeur de la grandeur à une itération donnée
- `visualization_tools.py` pour l'implémentation des fonctions nécessaires au calcul des grandeurs
- `histograms.py` pour l'extraction des données des fichiers annexes sous la forme de tableaux et la construction des histogrammes associés. Pour cela, nous avons 2 fonctions différentes pour tracer les histogrammes :
  - `histograms()` qui utilise les fonctions de Matplotlib `plt.hist()` en passant le tableau de données et `plt.show()` notamment. On sauvegarde ensuite la figure, nommée en fonction la date de création et de la grandeur mesurée, dans le dossier visualization_result
  - `dynamic_histograms()` qui utilise la fonction `go.Histogram()` de l'outil Plotly qui prend en paramètre le tableau de données et on affiche ensuite la figure sur une page web avec `fig.show()`.

Par ailleurs, pour la vérification de notre hypothèse, nous avons initialisé la valeur $\epsilon$ à $1 \times 10^{-10}$.

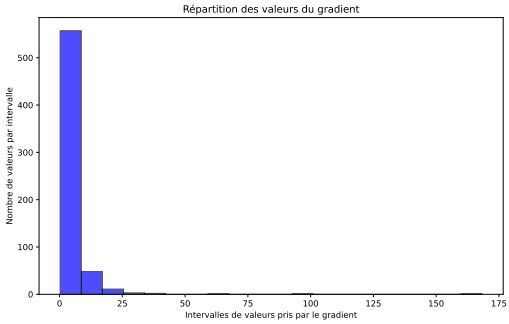
Pour nos tests, nous avons manipulé le *learning rate* : c'est-à-dire la taille du pas à chaque itération de l'algorithme vers la minimisation de la fonction de perte, dont on a testé les valeurs ($5 \times 10^{-2}$, $5 \times 10^{-3}$, $5 \times 10^{-5}$). Nos histogrammes comportent également le même nombre d'intervalles (20) car nous avons pensé qu'il s'agissait d'une valeur intéressante pour mieux voir la répartition des valeurs.

Nous obtenons donc les histogrammes suivants :

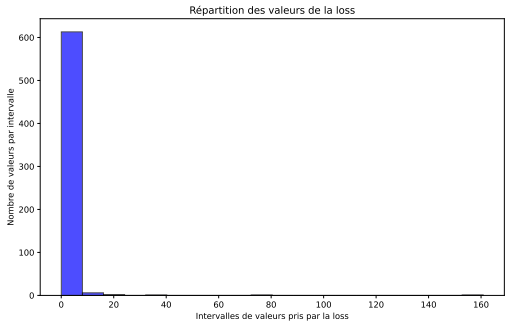
Pour un *learning rate* de $5 \times 10^{-2}$ :



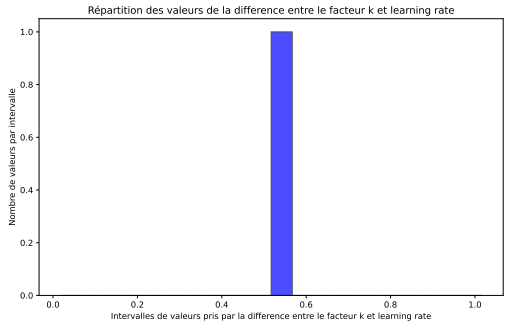

(a) distance




(b) gradient

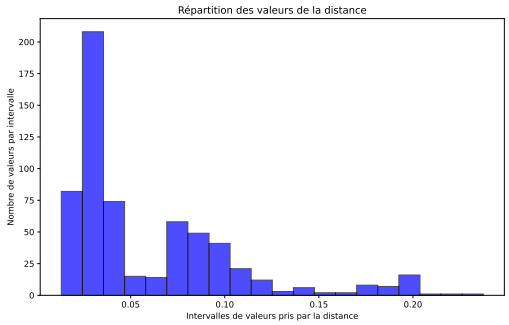



(c) *loss*

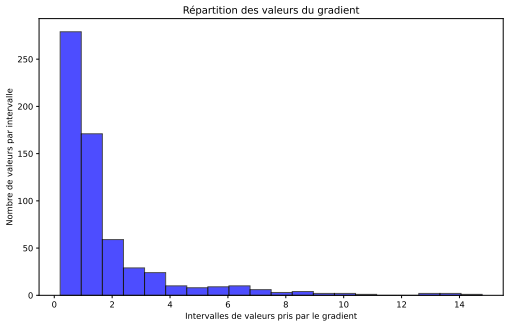



(d) différence

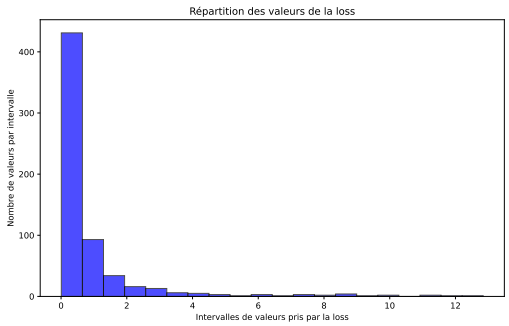
Pour un *learning rate* de $5 \times 10^{-3}$ :



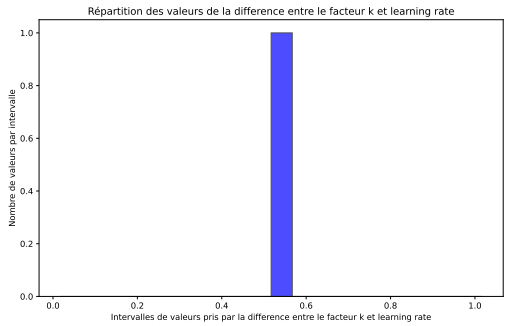

(a) distance




(b) gradient

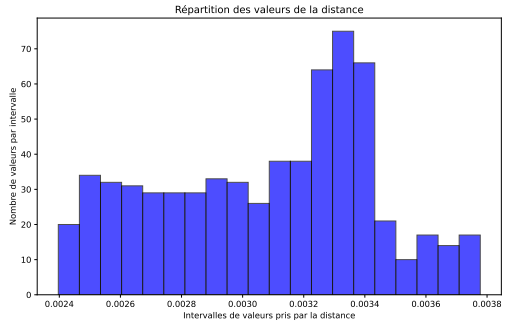



(c) *loss*

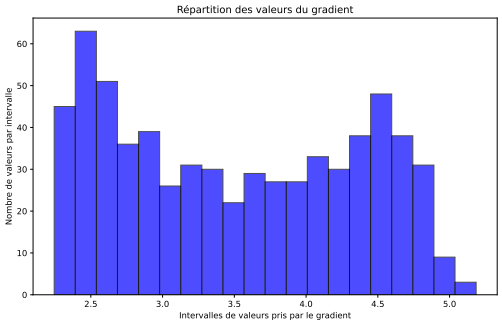



(d) différence

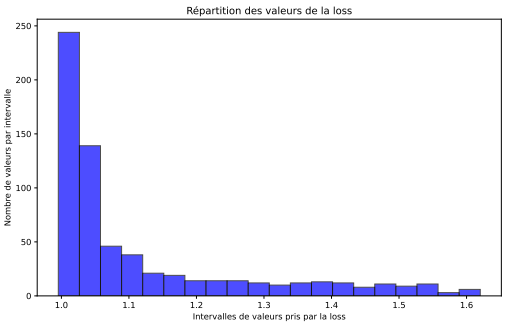
Pour un *learning rate* de $5 \times 10^{-5}$ :



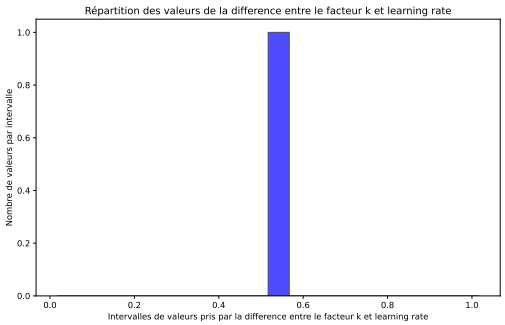

(a) distance




(b) gradient




(c) *loss*




(d) différence

Nous observons que l'hypothèse de départ n'est pas vérifiée pour une valeur $\epsilon$ valant $1 \times 10^{-10}$. En effet, la différence entre le facteur k calculée et le *learning rate* tourne davantage autour de valeurs proches de $1 \times 10^{-3}$ en général. Nous en concluons que l'hypothèse de départ est invalide dans nos expérimentations.

# 3 Visualisation des sous-espaces de l'espace de politiques

Dans cette section, nous présentons nos approches pour le développement de l'outil de visualisation de sous-espaces spécifiques définis au sein de l'espace global des politiques. Le code correspondant se trouve dans notre répertoire GitHub, le chemin vers le script étant : `src/bbrl_algos/visualization/policies_visualization.py`.

## 3.1 Récupération des meilleures politiques

Pour pouvoir construire un sous-espace de l'espace des politiques, nous devons d'abord choisir 3 politiques pour définir les extrémités de notre sous-espace, que nous allons appeler `P1, P2, P3`. Pour ce faire, nous avons utilisé le fait que les algorithmes de BBRL sauvegardent les agents donnant les meilleures politiques, en fonction de la récompense, dans des dossiers. Ainsi, il faut lancer un de ces algorithmes 3 fois, en utilisant des graines aléatoires différentes, afin d'obtenir une trace de ces politiques.

C'est, à présent, le moment où nous intervenons, avec l'aide des fonctions que nous avons développées, pour récupérer l'agent de la meilleure politique de chaque exécution. `get_best_policy(date, time, suffix, env_name_algo, algo, directory)` est la fonction qui nous permet de récupérer les agents donnant les meilleures politiques en fonction de la date, de l'heure de lancement de l'algorithme d'apprentissage, de l'environnement et de stocker l'agent dans le dossier `directory`. Un exemple de comment utiliser notre code est donné dans l'annexe.

Ensuite, à l'aide des fonctions `load_best_agent(directory)` et `load_policies(loaded_agents)`, nous allons donc récupérer les agents pour finalement récupérer les 3 meilleures politiques `P1, P2 et P3`.

## 3.2 Création du sous-espace de politiques

Maintenant, en ayant les politiques `P1`, `P2` et `P3`, nous allons pouvoir construire un sous-espace en forme de triangle, que nous avons choisi d'être équilatéral, défini avec les politiques `P1`, `P2` et `P3` comme extrémités du triangle. Ainsi, les coordonnées cartésiennes de nos politiques sont : `P1(0,0)`, `P2(1,0)` et `P3(1/2, √3/2)`. Le sous-espace est donc défini par les droites d'équation `y = 0*x`, `y = √3*x` et `y = √3*(1-x)`.

Ensuite, l'objectif est de remplir le triangle avec des politiques dérivées de nos politiques principales. Soit `P =` $\alpha_1$`P1+`$\alpha_2$`P2+`$\alpha_3$`P3` une politique dérivée. Sachant que, pour `P1(0,0)`, `P2(1,0)` et `P3(1/2, √3/2)`, les coefficients $\alpha$ correspondants sont $\alpha$=[1,0,0], $\alpha$=[0,1,0] et $\alpha$=[0,0,1] et que $\sum_{i=1}^{3} \alpha_i = 1$, nous devons maintenant remplir le sous-espace avec des politiques dérivées.

Pour l'instant, nous remplissons le triangle sans prendre en compte les valeurs des coefficients $\alpha$ et des nouvelles politiques dérivées. Nous générons d'abord des points sur les droites [P1 ;P2] et [P1 ;P3]. Comme c'est un triangle équilatéral, cela revient à générer des points sur les droites d'équation `y = 0*x` et `y = √3*x`.

Pour pouvoir remplir uniformément et efficacement le triangle, nous avons choisi de calculer l'intersection des perpendiculaires aux points que nous venons de générer. Tout

d'abord, nous savons que les médianes d'un triangle équilatéral sont également des hauteurs. Donc, pour la droite `y = 0`, la hauteur est donnée par `x = 1/2` et pour `y =` $\sqrt{3}$`*x` par `y =` $\sqrt{3}$`/3*(1-x)`. Ensuite, pour chaque point que nous avons généré, nous trouvons sa perpendiculaire qui est donc une droite parallèle à la hauteur `x = 1/2` ou à `y =` $\sqrt{3}$`/3*(1-x)`. Si la perpendiculaire est une droite parallèle à la hauteur alors les droites ont la même pente. En ayant donc les coordonnées de nos points générés et les pentes des hauteurs, nous pouvons donc calculer la perpendiculaire à chaque point généré. Puis, nous calculons l'intersection de nos perpendiculaires et nous remplissons donc le sous-espace de manière uniforme et efficace. Un exemple de comment remplir le sous-space est donné dans la Figure 4

De plus, lorsqu'un point de l'intersection ne se trouve pas dans l'espace délimité par le triangle, nous ne l'ajoutons pas dans le sous-espace pour éviter une évaluation inutile de la récompense. L'évaluation de la récompense est décrite dans la suite du rapport.

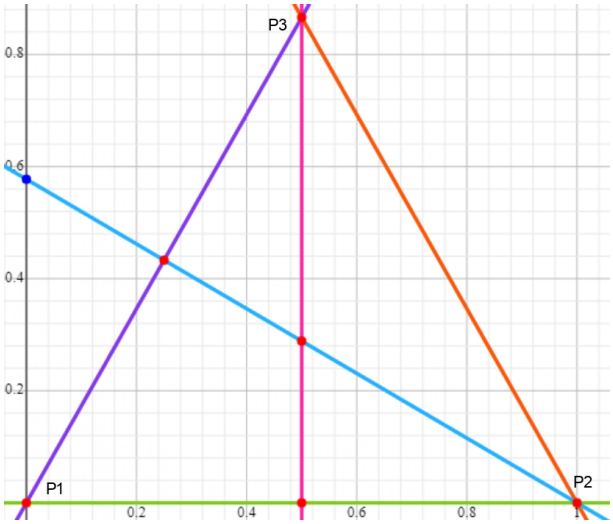



FIGURE 4 – Remplir le sous-space avec des points

Maintenant, il faut trouver les coefficients $\alpha$ associés aux points générés dans l'espace. Pour ce faire, nous devons donc résoudre le système linéaire suivant :

$$\begin{cases} x = \alpha_1 \cdot A_x + \alpha_2 \cdot B_x + \alpha_3 \cdot C_x \\ y = \alpha_1 \cdot A_y + \alpha_2 \cdot B_y + \alpha_3 \cdot C_y \\ \alpha_1 + \alpha_2 + \alpha_3 = 1 \\ \alpha_1, \alpha_2, \alpha_3 \geq 0 \end{cases}$$

où :

— $A_x, A_y$ sont les coordonnées du premier sommet du triangle (0,0).
— $B_x, B_y$ sont les coordonnées du deuxième sommet du triangle (1,0).
— $C_x, C_y$ sont les coordonnées du troisième sommet du triangle (1/2, $\sqrt{3}$/2).

et la fonction `get_alphas_from_point(x,y)` nous permet de résoudre ce système linéaire.

Avec les $\alpha$ connus, nous pouvons maintenant calculer la politique correspondante à chaque point dans le sous-espace, donnée par `P =` $\alpha_1$`P1+`$\alpha_2$`P2+`$\alpha_3$`P3`, et la fonction

`update_policy_with_coefficients(list_policies, coefficients)` est celle qui nous permet de calculer cette politique. L'objectif est d'évaluer cette nouvelle politique et de calculer sa récompense.

Pour pouvoir calculer sa récompense, nous devons d'abord créer un agent d'évaluation correspondant à l'algorithme d'apprentissage avec lequel nous avons obtenu nos 3 politiques initiales `P1, P2` et `P3`. Donc, dans notre cas, nous avons dû implémenter 2 fonctions `create_new_DQN_agent(cfg, env_agent)` et `create_new_TD3_agent(cfg, env_agent)`.

Ensuite, avec les agents d'évaluation créés, la fonction `evaluate_agent(eval_agent, theta)` nous permettra finalement de calculer la récompense de la politique `theta`, qui correspond à `P =` $\alpha_1$`P1+`$\alpha_2$`P2+`$\alpha_3$`P3`.

Finalement, `policies_visualization(eval_agent, num_points, loaded_policies, policies_traj, plot_traj)` est la fonction qui nous permet de tout mettre ensemble pour avoir toutes les données nécessaires pour la visualisation du sous-espace de politiques. Elle réalise tout le traitement que nous avons expliqué dans cette section, c'est-à-dire qu'elle remplit le triangle avec des points, calcule les coefficients $\alpha$ correspondants aux points, calcule les nouvelles politiques et les évalue. Cette fonction fait appel à une autre fonction, `plot_triangle_with_multiple_points(alpha_reward_list, alpha_reward_traj, plot_traj)`, qui permet de faire le tracé du sous-espace de politiques, en colorant chaque point qui représente une politique en fonction de sa récompense. Un exemple de visualisation est donné dans la Figure 5

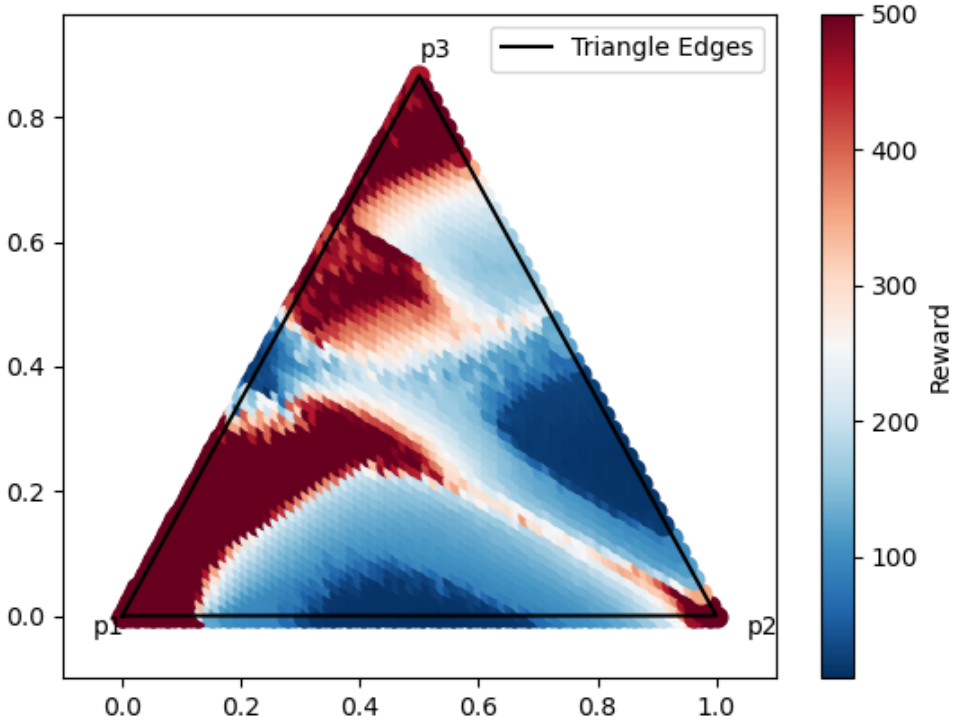



FIGURE 5 – Plot d'un sous-espace dans l'environnement Cartpole avec des couches cachées de taille *hidden_ sizes* : [64, 64]

À l'aide de la bibliothèque graphique Plotly, nous avons rendu notre tracé du sous-espace plus interactif. Nous avons implémenté une fonction adaptée à Plotly, sous le nom `plot_triangle_with_multiple_points_plotly(alpha_reward_list, alpha_reward_traj, plot_traj)`. Cette fonction nous permet de zoomer sur la figure et de voir les coordonnées de chaque point dans le plan, avec les $\alpha$ correspondants. Un exemple de visualisation sur Plotly est donné dans la Figure 7.

## 3.3 Trajectoire des politiques successives projetées dans le sous-espace

La fonction, `policies_visualization(eval_agent, num_points, loaded_policies, policies_traj, plot_traj)` est celle nous permettant d'obtenir le tracé du sous-espace des politiques. Cette fonction contient deux paramètres : `policies_traj`, une liste de politiques représentant la trajectoire des politiques dans l'apprentissage, et `plot_traj`, un booléen nous permettant d'afficher ou non cette trajectoire.

Pour garder une trace des politiques dans l'apprentissage des algorithmes, la fonction `save(agent, env_name, score, dirname, fileroot, cpt)`, enregistre dans le dossier `dirname` un certain nombre d'agents contenant les politiques dans l'algorithme d'apprentissage.

Donc, en connaissant notre politique `P` se trouvant dans la trajectoire, ainsi que nos politiques `P1`, `P2` et `P3` définissant le sous-espace, nous devons déterminer les coefficients $\alpha$ nous permettant de trouver cette politique `P`. Une fois de plus, un système linéaire doit être résolu :

$$\begin{cases} \mathrm{P} = \alpha_1 P_1 + \alpha_2 P_2 + \alpha_3 P_3 \\ \alpha_1 + \alpha_2 + \alpha_3 = 1 \\ \alpha_1, \alpha_2, \alpha_3 \geq 0 \end{cases}$$

et la fonction `projection_convex_hull(p, p1, p2, p3)` nous permet de le faire.

En ayant les coefficients $\alpha$ de nos politiques faisant partie de la trajectoire et comme nous sommes sûrs, grâce à la contrainte $\sum_{i=1}^{3} \alpha_i = 1$, que la trajectoire se trouve dans le sous-espace, il suffit donc de tracer la trajectoire également. Pour cela, un appel à `policies_visualization(eval_agent, num_points, loaded_policies, policies_traj, plot_traj)` avec `plot_traj` = *True* doit afficher le tracé avec la trajectoire.

Un exemple de tracé contenant la trajectoire peut être observé dans la Figure 6, où le début de la trajectoire est marqué en jaune, ainsi que la fin en vert. Nous tenons à préciser que le tracé de la trajectoire est également disponible dans sa version interactive avec Plotly, dans la Figure 7.

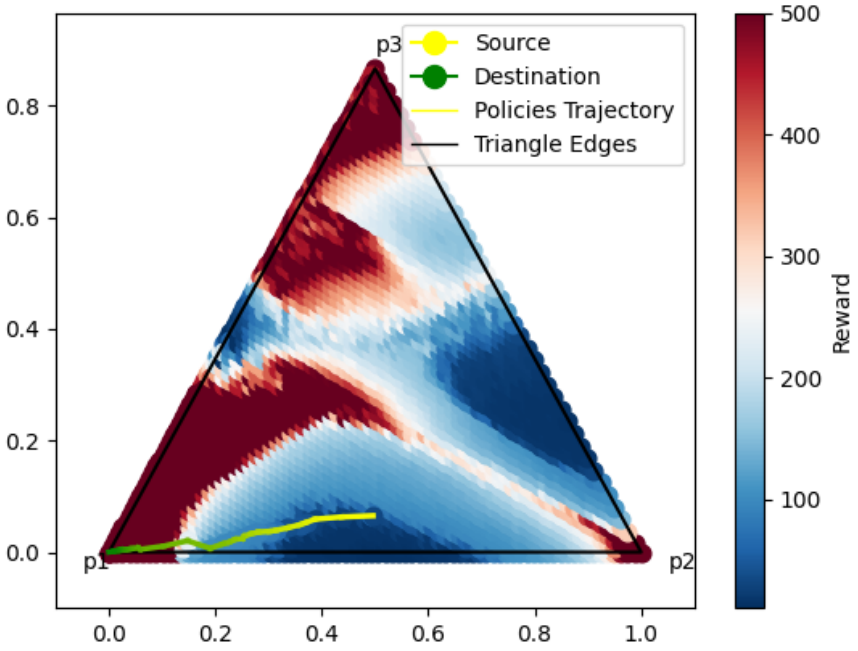



FIGURE 6 – Plot d'un sous-espace avec trajectoire dans l'environnement Cartpole avec des couches cachées de taille *hidden_ sizes* : [64, 64]

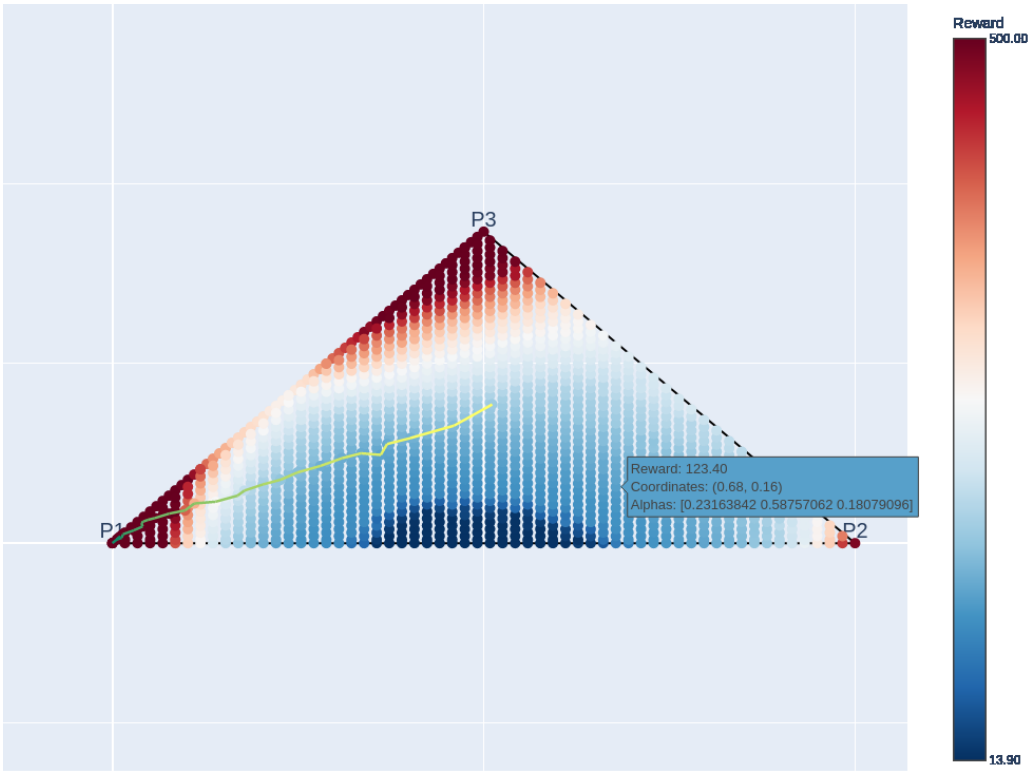



FIGURE 7 – Plot d'un sous-espace avec trajectoire sur Plotly dans l'environnement Cartpole avec des couches cachées de taille *hidden_ sizes* : [64, 64]

## 3.4 Conception d'un algorithme évolutif

Nous avons également élaboré un algorithme évolutif basé sur la méthode *cross-entropy method*. Notre objectif est de tester une alternative aux algorithmes d'apprentissage par renforcement.

Pour cela, nous prenons une population d'individus qui se déplacent aléatoirement dans le triangle défini par les trois meilleures politiques récupérées suite aux trois exécutions d'un algorithme d'apprentissage. Chaque individu se voit donc assigner un triplet $(\alpha_1, \alpha_2, \alpha_3)$ correspondant à ses coordonnées. Ensuite, nous calculons de nouvelles politiques associées à chaque triplet, en leur ajoutant un léger bruit afin d'explorer l'espace de recherche de manière plus diversifiée. Nous calculons la récompense associée à chaque politique, puis nous sélectionnons les trois politiques dont les récompenses sont les plus élevées pour mettre à jour les politiques déterminant les sommets du triangle. Ensuite, nous répétons le processsus jusqu'à obtenir des résultats concluants.

Notre algorithme nous permet également de garder la trajectoire de la population de l'algorithme en mémorisant la politique la plus performante pendant chaque itération. En ayant cela, nous pouvons donc faire une comparaison entre la trajectoire d'un algorithme d'apprentissage par renforcement (Figure 6) et la trajectoire d'un algorithme évolutif (Figure 8).

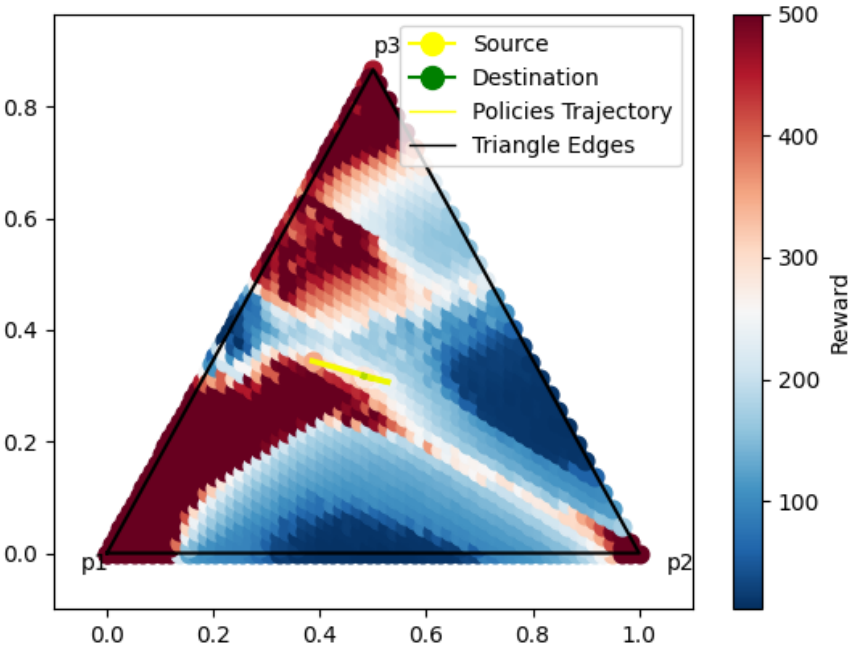



FIGURE 8 – Plot avec trajectoire obtenu en utilisant l'aglorithme évolutif avec une population de 20 individus

Comme attendu, la trajectoire lors de l'utilisation d'un algorithme d'apprentissage par renforcement semble avoir un objectif clair, ce dernier étant de se diriger vers une politique optimale. Cependant, pour l'algorithme évolutif, étant donné que l'on introduit une part d'aléatoire dans la génération de la population, et que pour chaque itération,

nous reconstruisons le sous-espace en sélectionnant les 3 meilleures politiques, notre sous-espace se restreint inévitablement. De plus, comme la trajectoire est représentée dans le sous-espace défini par les 3 meilleures politiques passées en paramètre de l'algorithme évolutif, la trajectoire devient difficile à suivre en raison de la réduction du sous-espace.

C'est maintenant que notre outil de visualisation, développé à l'aide de Plotly, intervient, car c'est lui qui nous permet d'explorer le sous-espace. En faisant un zoom sur la figure, nous pouvons voir plus clairement la trajectoire. Le zoom est représenté dans la Figure 9.

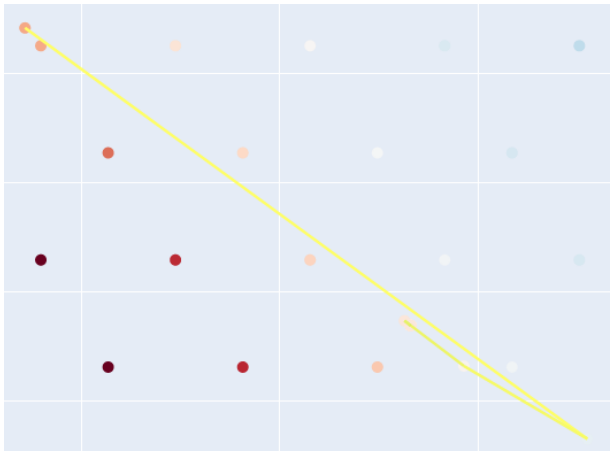


FIGURE 9 – Zoom sur la trajectoire de la Figure 8

Le point de destination (en vert) de la trajectoire n'est pas encore visible, donc il faut agrandir davantage l'image pour pouvoir le trouver.


FIGURE 10 – Zoom sur la Figure 9

# 4 Bibliographie

[1] Building a subspace of policies for scalable continual learning.

https://arxiv.org/pdf/2211.10445.pdf

Autheurs : Jean-Baptiste Gaya, Thang Doan, Lucas Caccia, Laure Soulier, Ludovic Denoyer, Roberta Raileanu

[2] Learning a subspace of policies for online adaptation in reinforcement learning

https://arxiv.org/pdf/2110.05169.pdf

Autheurs : Jean-Baptiste Gaya, Laure Soulier, Ludovic Denoyer

# 5 Annexe

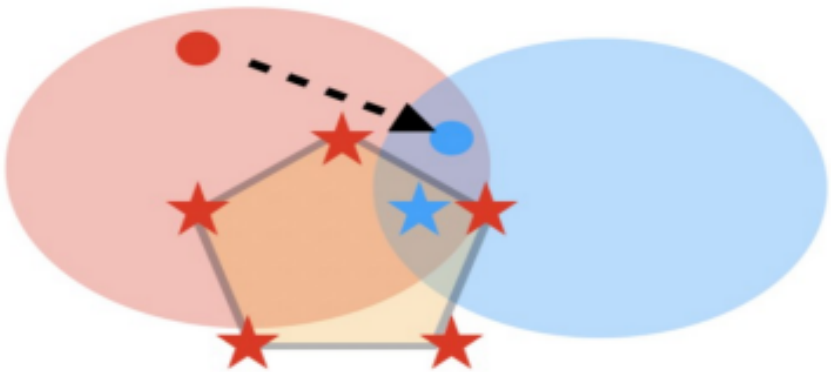
## Cahier de charges

### Introduction et utilité

Ce projet se concentre sur l'étude des espaces des politiques, définis comme l'ensemble généré en faisant varier les paramètres d'un réseau de neurones correspondant à une politique, soit l'entité qui contrôle le comportement d'un agent. Des recherches préalables ont démontré qu'une sous-partie spécifique de cet espace, délimitée par un hyper-plan traversant trois politiques performantes et appelée "subspace", contient elle-même des politiques performantes.

L'intérêt majeur d'un tel hyper-plan réside dans sa visualisation aisée en deux dimensions, permettant ainsi d'observer la distribution des politiques performantes dans le plan correspondant. Un exemple est fourni dans la Figure 11

FIGURE 11



Le schéma représente l'espace des paramètres. La région rouge (respectivement bleue) est l'espace des bonnes politiques sur l'environnement d'entraînement (respectivement de test). Une seule politique apprise (point rouge) peut être inefficace pour l'environnement de test et doit être adaptée (par exemple, affinée) pour devenir bonne au moment du test (point bleu). Au lieu d'apprendre une seule politique, nous apprenons un sous-espace convexe (le pentagone) délimité par des politiques d'ancrage (étoiles rouges) qui vise à capturer un grand ensemble de bonnes politiques. Ensuite, l'adaptation est simplement réalisée en échantillonnant des politiques dans ce sous-espace, en gardant la meilleure (étoile bleue).

Le premier objectif de ce projet consistera à élaborer un outil de visualisation spécifique à cet effet. Dans une phase ultérieure, le projet pourrait évoluer soit vers le développement d'outils de visualisation complémentaires, tels que la trajectoire suivie par un algorithme

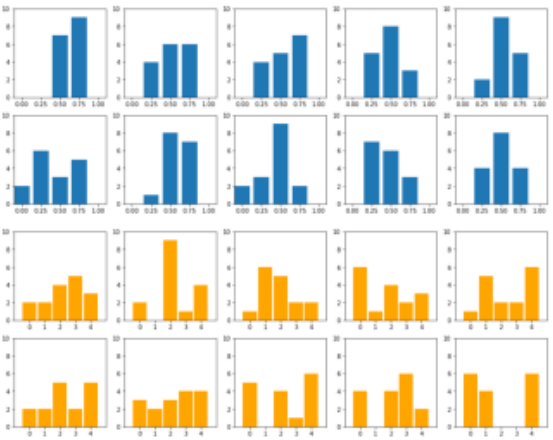
d'apprentissage dans l'espace des politiques, soit vers la création d'une méthode évolutive d'optimisation de politique fondée sur la propriété centrale des subspaces.

L'outil sera initialement adapté pour une étude des pas de gradient en apprentissage par renforcement profond, réalisée au sein d'algorithmes exploitant la descente du gradient. Dans le cadre de ce projet, des modifications seront apportées à la bibliothèque "bbrl_algos" de notre encadrant, en particulier dans le code de l'algorithme DQN (Deep Q-Network), qui est une méthode d'apprentissage par renforcement sans modèle (model free), en ligne (online), hors politique (off-policy). Un agent DQN est un agent d'apprentissage par renforcement basé sur la valeur, formant un critique pour estimer la récompense cumulée à long terme escomptée. DQN constitue une variante de Q-learning. Ensuite, nous allons adapter notre outil pour tous les algorithmes de la bibliothèque "bbrl_algos".

### Histogrammes

Dans la phase initiale de notre projet, nous nous focalisons sur l'analyse de la perte (*loss*), de la norme des distances entre deux politiques et de la norme des gradients dans le cadre de l'algorithme de la descente du gradient, spécifiquement avec l'utilisation de DQN. Pour cela, nous utilisons des histogrammes pour représenter ces éléments, s'inspirant de ceux présentés dans la Figure 12.

FIGURE 12



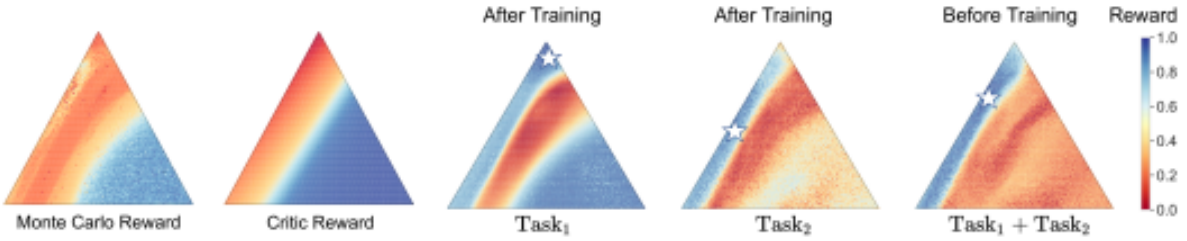
Ensuite, ces histogrammes deviennent plus dynamiques grâce aux outils de visualisation, permettant par exemple de zoomer sur la figure pour un examen plus détaillé.

Grâce à ces histogrammes ainsi qu'à nos fonctions de calcul des normes, nous pourrons vérifier certaines hypothèses, comme celle selon laquelle la norme du gradient est proportionnelle à la distance entre deux politiques successives, et nous pourrons vérifier si cette proportionnalité est déterminée par le taux d'apprentissage.

### Sous-espaces de l'espace de politiques

Dans cette section de notre projet, nous allons élaborer un outil de visualisation nous permettant d'afficher les performances des politiques dans un sous-espace sélectionné de l'espace des politiques. Un exemple de cette visualisation est présenté dans la Figure 13, ou en regardant le lien suivant : https://continual-subspace-policies-streamlit-app-gofujp.streamlit.app/Visualizing_the_Subspaces.

FIGURE 13




L'objectif final de notre projet sera donc que notre outil soit capable d'afficher quelque chose de similaire, mais en plus, il sera capable de représenter la trajectoire des politiques successives projetées dans le sous-espace. À l'aide de ces représentations, une étude sera menée sur la manière dont la trajectoire des politiques évolue lorsque différents hyperparamètres de notre modèle sont modifiés.

## Manuel de l'utilisateur

### Histogrammes

Pour pouvoir obtenir des histogrammes, il suffit de se diriger dans le dossier `visualization`, ayant le chemin : `src/bbrl_algos/visualization`, et d'exécuter `histograms.py`.

Le script `histograms.py` lance une exécution de l'algorithme DQN. À la fin de l'exécution de l'algorithme, en faisant appel à `histograms(nom_fichier.txt)`, notre code fait des illustrations en fonction du fichier texte passé en paramètre. De plus, la fonction `dynamic_histograms` permet de faire la même chose, mais en utilisant Plotly.

### Visualisation d'un sous-espace de l'espace de politiques

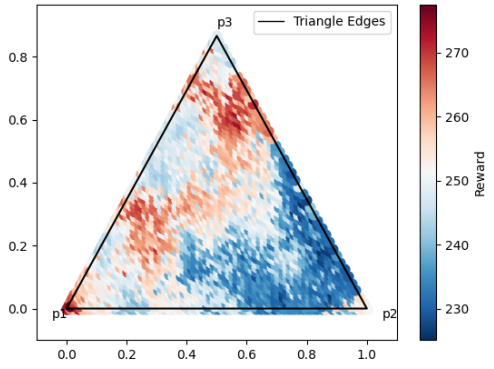
Pour visualiser un sous-espace de politiques, le script `policies_visualization.py` situé dans le dossier `src/bbrl_algos/visualization` doit être lancé. Avant l'appel, des appels à la fonction `get_best_policy` doivent être effectués pour récupérer les agents des 3 meilleures politiques en fonction de la date, de l'environnement et de l'algorithme. Il faut ensuite charger les agents du dossier spécifié dans `get_best_policy` en utilisant la fonction `load_best_agent`, puis de créer un agent d'évaluation en utilisant les fonctions `local_get_env_agents` et `create_new_DQN_agent` ou `create_new_TD3_agent` selon le choix de l'algorithme. Enfin, il faut faire appel à la fonction `policies_visualization` avec le nombre de points souhaité dans le sous-espace, initialement sans la trajectoire.

Si nous voulons également visualiser la trajectoire, les fonctions `read_and_sort_agents` et `load_policies` permettent de récupérer les politiques et de fournir la trajectoire d'apprentissage. Ensuite, un appel à `policies_visualization` doit être effectué, cette fois-ci en passant en paramètre la liste de trajectoire et en mettant le booléen `plot_traj` à *True*.

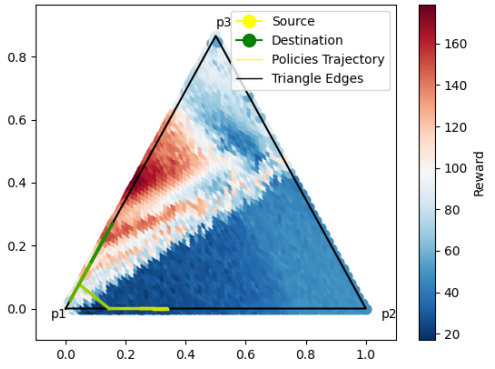
Pour l'algorithme évolutif, un appel à `evo_algo` suffit en mettant les 3 meilleures politiques en paramètre.

De plus, pour changer l'algorithme et l'environnement, il suffit de changer les variables `config_path` et `config_name` du `main`, et de passer l'algorithme et l'environnement souhaités dans l'appel de la fonction `get_best_policy`. Toutes les possibilités sont mises en commentaire dans le `main` du script `policies_visualization.py`.

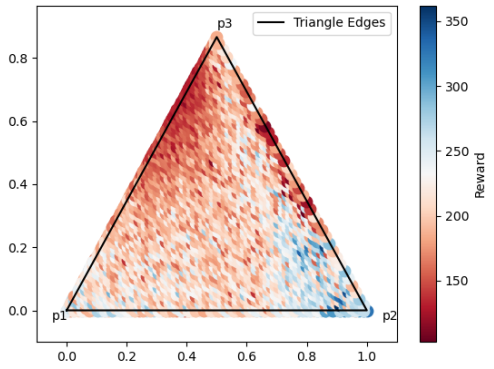
## Résultats complémentaires



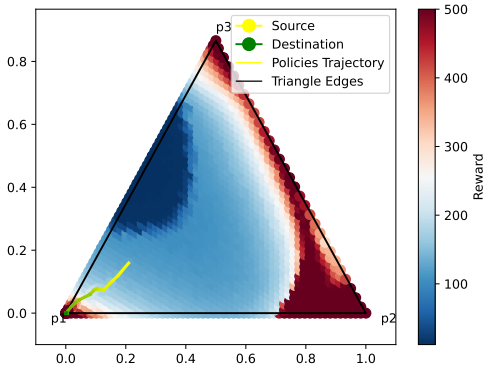

(a) Plot d'un sous-espace dans l'environnement Swimmer avec des couches cachées de taille *actor_hidden_size* : [64, 64] et *critic_hidden_size* : [512, 512]



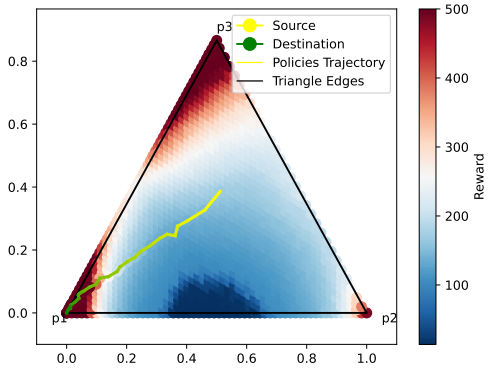

(b) Plot d'un sous-espace avec trajectoire dans l'environnement Swimmer avec des couches cachées de taille *actor_hidden_size* : [207, 207] et *critic_hidden_size* : [256, 256]



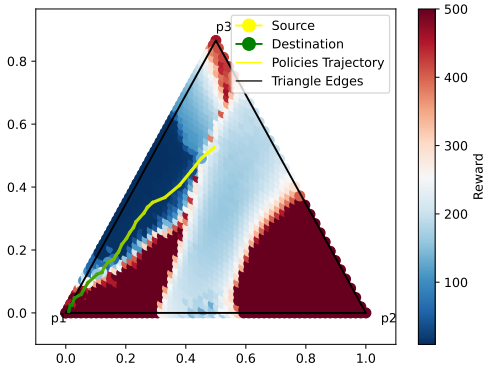

(c) Plot d'un sous-espace dans l'environnement Swimmer avec des couches cachées de taille *actor_hidden_size* : [64, 64] et *critic_hidden_size* : [512, 512]




(d) Plot d'un sous-espace avec trajectoire dans l'environnement Cartpole avec de couches cachées de taille *hidden_sizes* : [64, 64]




(e) Plot d'un sous-espace avec trajectoire dans l'environnement Cartpole avec de couches cachées de taille *hidden_sizes* : [64, 64]




(f) Plot d'un sous-espace avec trajectoire dans l'environnement Cartpole avec de couches cachées de taille *hidden_sizes* : [32, 32]