Master 2

RLD
Compte Rendu du Mini-projet

Binôme
Hatem AYED (DAC)
Abdelkrim HALIMI (DAC)

4 octobre 2024

Table des matières

I Introduction

L'objectif de ce mini-projet est de mettre en œuvre et d'analyser un algorithme actor-critic appliqué à un environnement de type MDP (Markov Decision Process) où les fonctions de transition et de récompense sont inconnues. Le projet vise à optimiser les hyperparamètres α_{critic} et α_{actor} afin d'améliorer le temps de convergence de l'algorithme. Des tests statistiques sont utilisés pour comparer les différentes configurations.
Notre analyse portera d'abord sur le cas où l'environnement est fixe. Par la suite, nous généraliserons le problème à l'optimisation dans un labyrinthe quelconque (de taille fixée cependant).

II Algorithme Actor-Critic

Nous avons implémenté l'algorithme *Actor-Critic* en nous appuyant sur les principes des algorithmes Q-Learning et Sarsa vus dans les TP précédents.
Cependant, l'algorithme Actor-Critic se distingue par son approche en deux parties :

— **Critic** : Le *critic* évalue les états en mettant à jour la fonction de valeur $V(s)$ à l'aide de l'erreur de différence temporelle (TD) :

$$\delta_t = r(s_t, a_t) + \gamma V(s_{t+1}) - V(s_t)$$

Ensuite, la mise à jour de la fonction de valeur se fait ainsi :

$$V(s_t) \leftarrow V(s_t) + \alpha_{critic} \cdot \delta_t$$

— **Actor** : L'*actor* ajuste la politique en fonction de l'erreur TD δ_t avec la mise à jour suivante :

$$\pi(a_t|s_t) \leftarrow \pi(a_t|s_t) + \alpha_{actor} \cdot \delta_t$$

Une renormalisation est ensuite appliquée pour garantir que les probabilités restent valides.

En Python, les modifications apportées sont simples. La Q-table est "remplacée" par une matrice des probabilités d'actions conditionnées aux états :

```
a_probs = np.ones((mdp.unwrapped.nb_states, mdp.action_space.n))/mdp.action_space.n
```

Les mises à jour sont effectuées de manière à éviter les probabilités négatives :

```
delta = r + mdp.gamma*v[y]*(1-terminated) - v[x]
v[x] = v[x] + alpha_critic*delta
temp = a_probs[x,u] + alpha_actor*delta
a_probs[x, u] = temp if temp > 1e-8 else 1e-8
a_probs[x] = a_probs[x]/a_probs[x].sum()
```

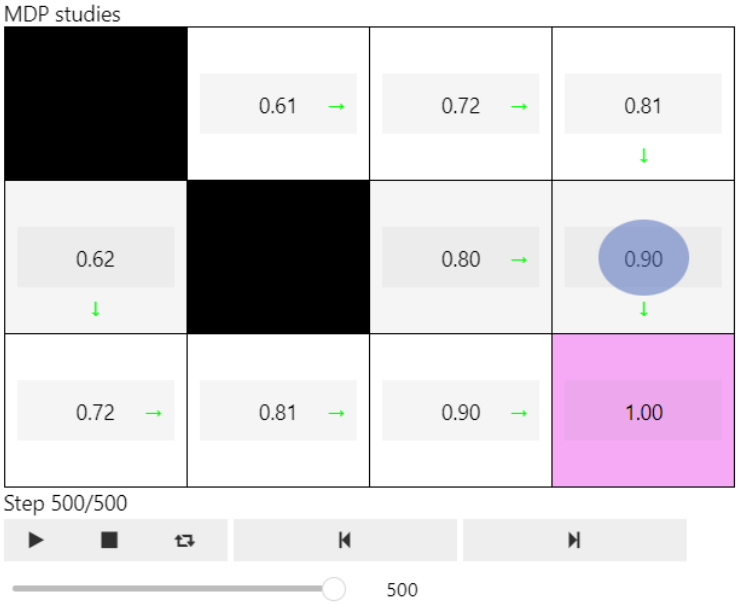

FIGURE 1 – Exemple d'exécution

Test de l'algorithme

Nous avons exécuté notre algorithme 15 fois. Comme le montre la Figure 2, une politique de plus en plus efficace est apprise au fil du temps. L'algorithme semble converger dès le 40e épisode. En effet, le nombre moyen de pas pour sortir du labyrinthe oscille autour de 5, avec une variance relativement stable.

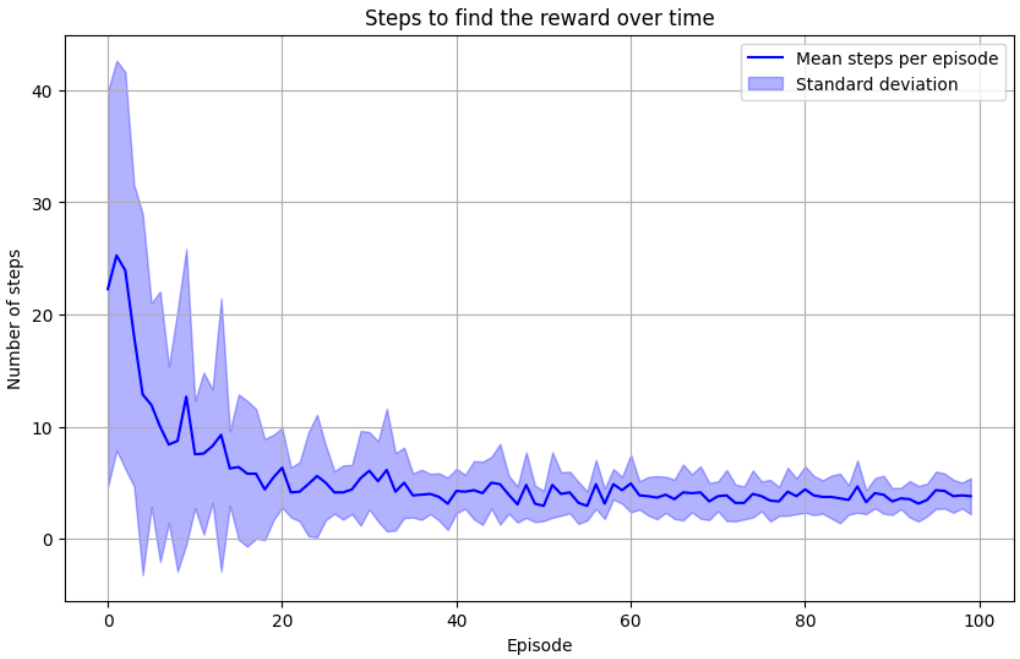

FIGURE 2 – Nombre de pas moyen et écart-type sur 15 exécutions

Notre algorithme est donc fonctionnel. Nous allons maintenant chercher à l'optimiser en déterminant les meilleurs hyperparamètres.

III Recherche des hyperparamètres optimaux

L'algorithme dispose de deux hyperparamètres principaux :
— α_{actor}, représentant le taux d'apprentissage de l'acteur
— α_{critic}, représentant le taux d'apprentissage du critique
Dans un premier temps, nous effectuerons nos recherches sur un labyrinthe fixe de taille 5x5, en utilisant notamment Optuna pour optimiser ces hyperparamètres.

III.1 Recherche sur un seul environnement

Nous allons explorer deux méthodes d'optimisation : Gridsearch et l'optimisation bayésienne avec Optuna, qui viseront à optimiser le nombre moyen de pas lors d'une session d'entraînement.

III.1.1 Optimisation via Gridsearch

La méthode de *Gridsearch* consiste à explorer une grille de valeurs prédéfinies pour les hyperparamètres. Nous avons choisi de tester 20 valeurs uniformément réparties entre 0.01 et 1 pour chaque hyperparamètre.

Pour chaque paire $(\alpha_{actor}^{(i)}, \alpha_{critic}^{(j)})$, nous réalisons $n_{repet} = 5$ exécutions indépendantes de l'algorithme , et collectons le nombre de pas nécessaires pour atteindre l'objectif, noté $t_k^{(i,j)}$.

La procédure d'évaluation est la suivante :

1. Nous exécutons $n_{episodes} = 150$ épisodes pour chaque répétition, collectant une séquence de $t_k^{(i,j)}$ pour chaque paire (i, j).
2. Pour chaque répétition r (avec $r = 1, \ldots, n_{repet}$), nous calculons le nombre moyen de pas par épisode :
$$\bar{t}^{(i,j,r)} = \frac{1}{n_{episodes}} \sum_{k=1}^{n_{episodes}} t_k^{(i,j,r)}$$
3. Ensuite, nous calculons la moyenne des résultats obtenus sur les n_{repet} répétitions :
$$\bar{t}^{(i,j)} = \frac{1}{n_{repet}} \sum_{r=1}^{n_{repet}} \bar{t}^{(i,j,r)}$$
4. Finalement, les indices (i^*, j^*) qui minimisent la moyenne des pas sont définis comme :
$$(i^*, j^*) = \text{argmin}_{(i,j)}\, \bar{t}^{(i,j)}$$
La paire d'hyperparamètres optimale est alors donnée par $(\alpha_{actor}^{(i^*)}, \alpha_{critic}^{(j^*)})$.

Les résultats de cette exploration sont visualisés sous la forme d'une heatmap (Figure 3) où chaque paire d'hyperparamètres $(\alpha_{actor}^{(i)}, \alpha_{critic}^{(j)})$ est associée à la valeur de $\bar{t}^{(i,j)}$.
Les meilleurs hyperparamètres trouvés sont : $\alpha_{actor} = 1.0$, $\alpha_{critic} = 0.635$, avec une moyenne de 10 pas.

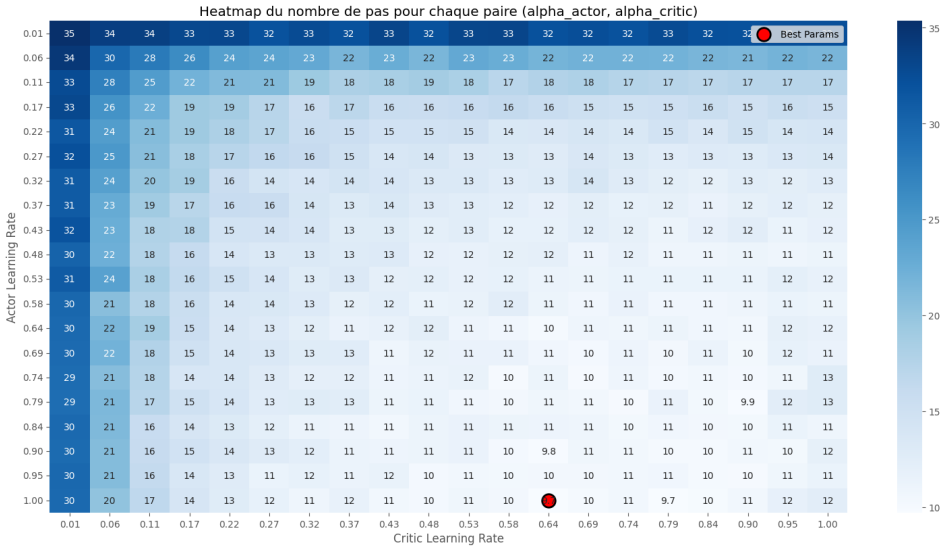

FIGURE 3 – Heatmap des hyperparamètres

III.1.2 Optimisation bayésienne

Nous utilisons à présent une méthode d'optimisation bayésienne pour déterminer les hyperparamètres optimaux. Comme dans la section précédente, la fonction à minimiser est le nombre de pas moyen pour sortir du labyrinthe, tel que défini section III.1.1.

Les meilleurs hyperparamètres trouvés sont : α_{actor} : 0.967, α_{critic} : 0.793, donnant 9 pas en moyenne.

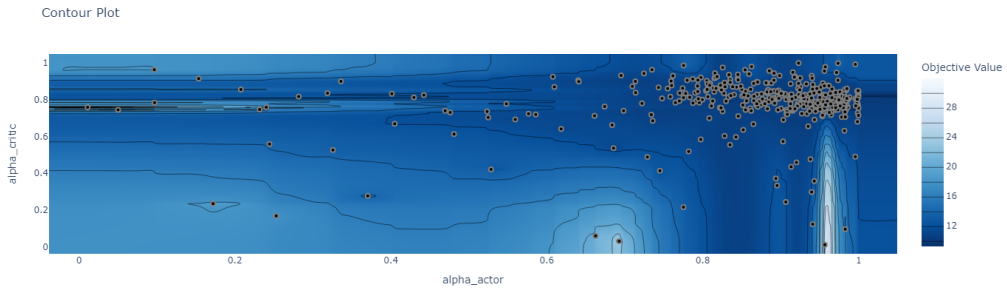

FIGURE 4 – Recherche bayésienne avec Optuna

Comme nous pouvons l'observer sur la figure ci-dessus, les hyperparamètres explorés forment un groupe concentré autour d'une zone d'intérêt. Cette distribution met en évidence l'une des principales différences entre *Gridsearch* et l'optimisation bayésienne. Alors que la première explore l'espace des hyperparamètres de manière exhaustive et uniforme, l'optimisation bayésienne

affine progressivement ses recherches en se basant sur les résultats précédents, ce qui permet une exploration plus localisée et efficace.

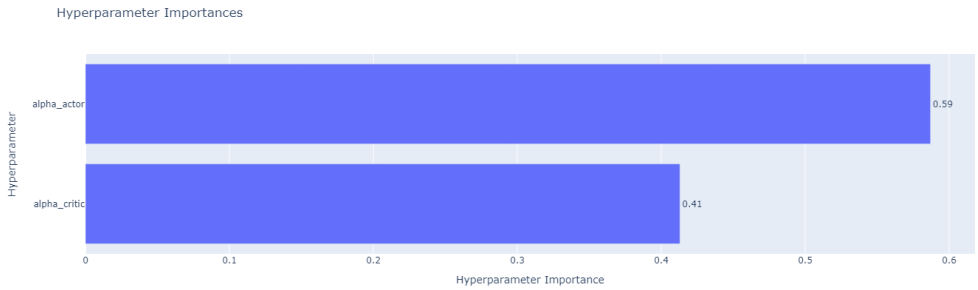

FIGURE 5 – Importance des hyperparamètres

La figure ci-dessus présente l'importance relative des hyperparamètres. Nous remarquons que le paramètre α_{actor} a une influence plus marquée sur les performances du modèle par rapport à α_{critic}, confirmant que la recherche des paramètres de manière à ce que ceux à $t+1$ dépendent des informations collectées à t.

III.2 Test statistique sur 1 seul environnement

Après avoir identifié les hyperparamètres optimaux, il est essentiel de les évaluer. Pour cela, nous effectuons un test de Welch afin de détecter les différences significatives entre les courbes de performance (signalées par des points noirs). La figure ci-dessous montre l'évolution des performances de l'algorithme (nombre moyen de pas) pour 15 répétitions de 150 épisodes. Nous comparons les résultats obtenus avec les hyperparamètres par défaut (0.5, 0.5) aux hyperparamètres optimisés trouvés à l'aide de la recherche bayésienne et du Gridsearch.

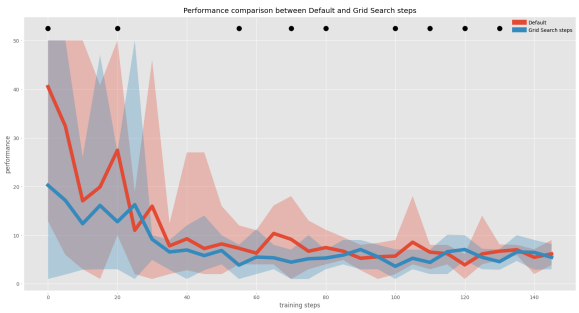

FIGURE 6 – Comparaison entre Grid Search et Naive

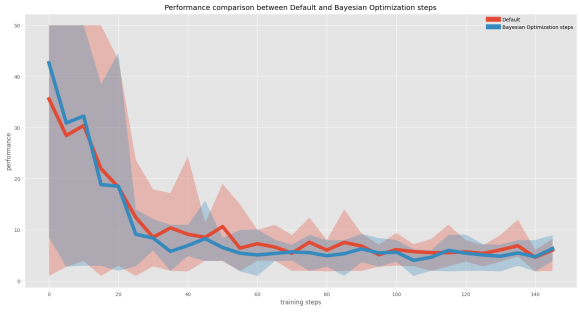

FIGURE 7 – Comparaison entre Optimisation Bayésienne et Naive

On constate que les hyperparamètres trouvés sont significativement meilleurs que les paramètres par défaut (dans le cas de GridSearch). Cependant, il est important de noter que nous avons toujours travaillé sur le même environnement, à savoir le même labyrinthe. Bien que nos hyperparamètres soient supérieurs dans ce cas précis, permettent-ils une certaine généralisation ?

III.2.1 Examen des performances sur différents labyrinthes

Nous générons 100 labyrinthes différents et comparons les performances entre notre paire d'hyperparamètres naïve et celle trouvée précédemment.

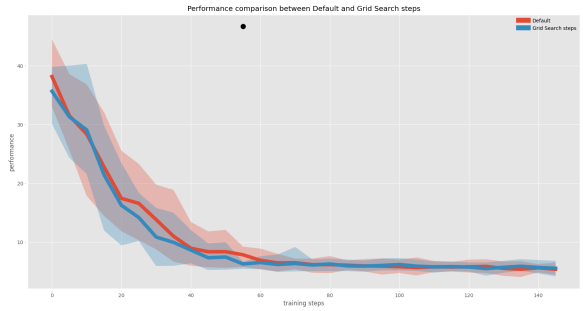

FIGURE 8 – Comparaison entre Grid Search et Naive

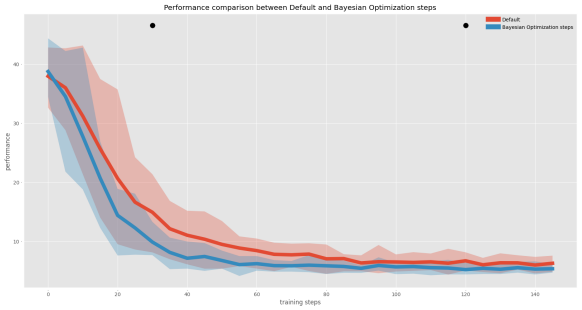

FIGURE 9 – Comparaison entre Optimisation Bayésienne et Naive

FIGURE 10 – Ensemble de test sur 100 environnements

Sans surprise, on constate qu'il n'y a plus de différence significative : nos hyperparamètres étaient optimaux uniquement pour un labyrinthe donné. Cela nous amène à notre prochain objectif : trouver une paire d'hyperparamètres qui permettrait de généraliser, afin d'obtenir les meilleures performances possibles sur n'importe quel labyrinthe de taille 5x5.

III.3 Recherche sur plusieurs environnements

Dans cette partie, nous allons construire une base d'entraînement constituée de 5 environnements distincts sur laquelle nous allons optimiser nos hyperparamètres. Le but sera de voir si sur un labyrinthe aléatoire donné, nos nouveaux hyperparamètres seront en moyenne plus performants que ceux trouvés dans la partie précédente. Pour ce faire, nous avons choisi les paramètres suivants :

— Nombre d'environnements : 5
— Nombre de répétitions de l'algorithme : $n_{repet} = 15$
— Fonction à minimiser : Nombre de pas moyens

Comme dans le cas avec un seul environnement, nous avons effectué les mêmes types de recherche d'hyperparamètres (GridSearch et Bayesian Optimisation).

III.3.1 Optimisation via GridSearch et Bayesian Optimisation

La paire optimale trouvée sur nos 5 environnements fixés via GridSearch est la suivante :
— α_{actor} : 1.0
— α_{critic} : 0.844

La paire trouvée via Bayesian Optimisation est la suivante :
— α_{actor} : 0.988
— α_{critic} : 0.804

On constate que dans les deux cas, le learning rate du critic est inférieur à celui de l'acteur, ce qui n'est pas courant dans la littérature. Cela peut être dû au fait que nous optimisons le nombre de pas sur un nombre limité d'environnements.

III.4 Test statistique - Plusieurs environnements

III.4.1 Tests sur les 5 environnements d'entraînement

Commençons par comparer les performances des 3 paires d'hyperparamètres sur les 5 environnements d'entraînement.

— **Valeurs par défaut vs Bayesian Optimisation**

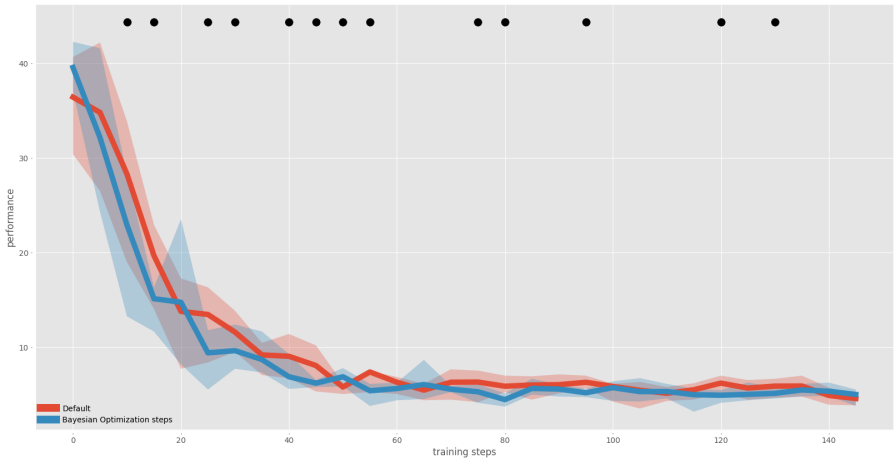

FIGURE 11 – Nombre de pas moyens par épisode : Valeurs par défaut vs Bayesian Optimisation

Sur le graphe du nombre de pas (Figure 11), on constate des différences significatives au début de l'entraînement, différences qui se réduisent au cours des épisodes jusqu'a convergence.

— **Valeurs par défaut vs GridSearch**

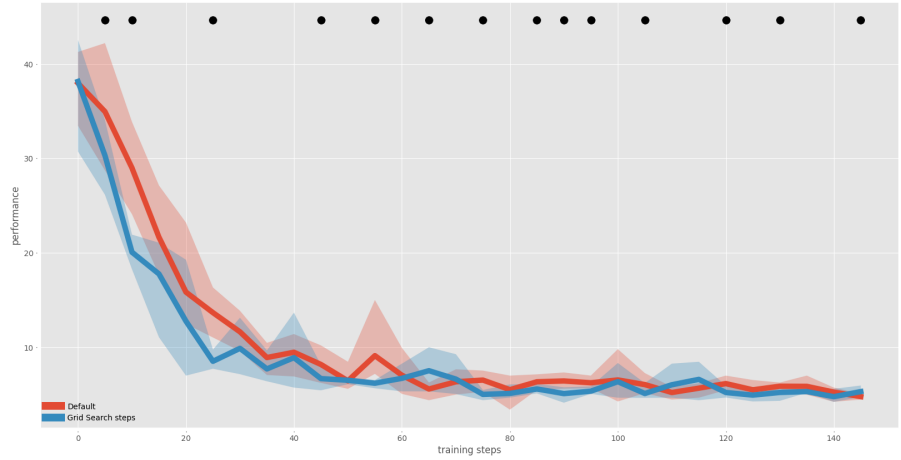

FIGURE 12 – Nombre de pas moyens par épisode : Valeurs par défaut vs GridSearch

En comparant avec GridSearch, on constate que les différences restent significatives sur le long terme, bien qu'elles finissent par se réduire lors des derniers épisodes.

— **Valeurs de GridSearch vs BayesianOptimisation**

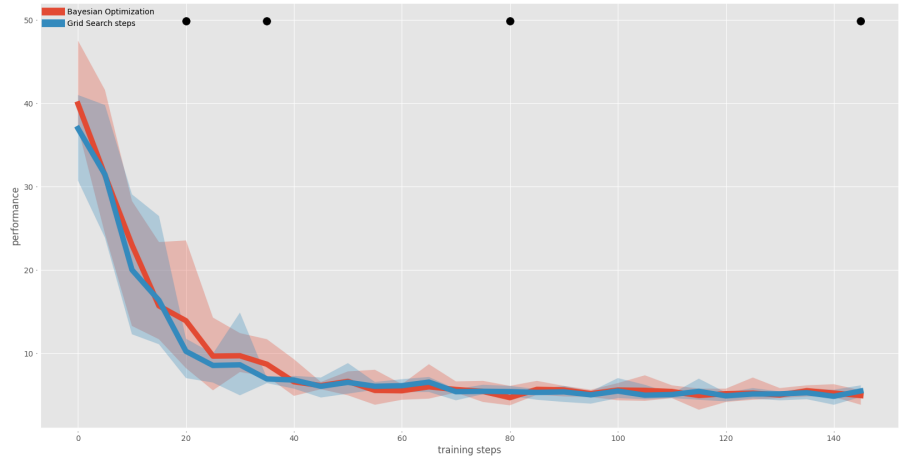

FIGURE 13 – Nombre de pas moyens par épisode : GridSearch vs BayesianOptimisation

Enfin, pour les meilleurs hyperparamètres trouvés par les deux algorithmes de recherche, on constate des résultats similaires en termes de pas. Bien que l'on observe deux différences significatives au début, montrant que les paramètres trouvés par GridSearch convergent plus vite à 20 et 37 épisodes, et qu'une différence significative soit atteinte au dernier épisode avec l'algorithme Bayesian Optimisation, les deux meilleurs hyperparamètres trouvés par les deux algorithmes semblent équivalents de manière globale.

III.4.2 Tests sur des environnements aléatoires

Testons maintenant nos nouveaux hyperparamètres sur 100 environnements aléatoires pour voir si nous avons atteint notre objectif de généralisation via notre méthode.

Pour cela, nous allons comparer les meilleurs paramètres obtenus avec l'optimisation Bayésienne, étant donné que cette méthode donne quasiment les mêmes résultats que la méthode Grid Search d'après nos tests, avec les paramètres par défaut :

— α_{actor} : 0.5
— α_{critic} : 0.5

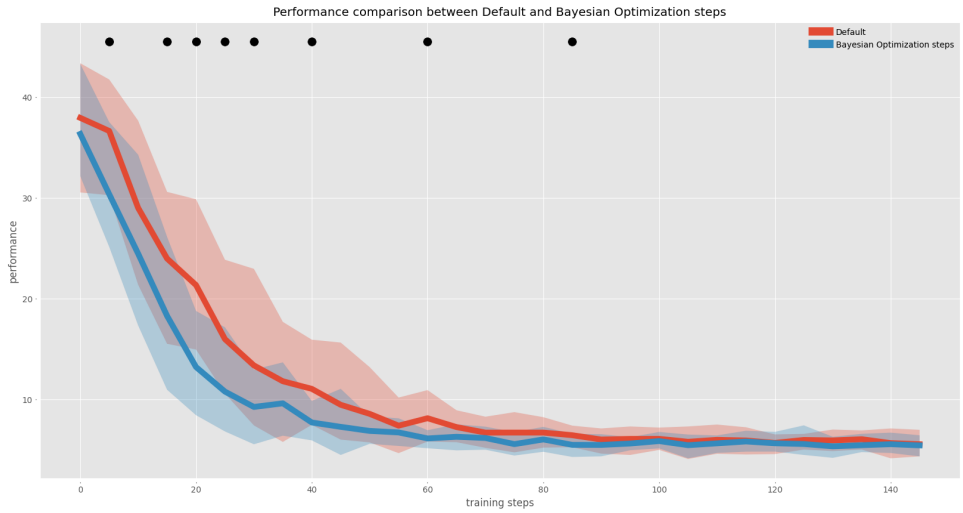

FIGURE 14 – Résultat sur 100 environnements aléatoires

Comme le montre la figure, les paramètres par défaut convergent bien moins rapidement de manière significative au début par rapport à nos paramètres optimaux, qui ont été entraînés sur 5 environnements. Ainsi, l'augmentation des environnements a permis une certaine généralisation pour notre méthode.

IV Maximisation de la norme vs Minimisation du nombre de pas

Nous avons voulu comparer les différences des hyperparamètres trouvés par les deux algorithmes en utilisant deux métriques différentes, à savoir la norme du vecteur V et le nombre de pas moyens. Pour cela, nous avons effectué d'autres tests statistiques sur les résultats obtenus. Meilleurs paramètres trouvés en maximisant la norme

— α_{actor}, 0.841
— α_{critic}, 0.861

On constate déjà que le résultat montre un α_{critic} plus élevé que α_{actor} (cas inverse en minimisant les pas moyens).

Ci-dessous, des tests statistiques pour comparer les deux approches :

— **GridSearch (Max norm) vs (Min steps)**

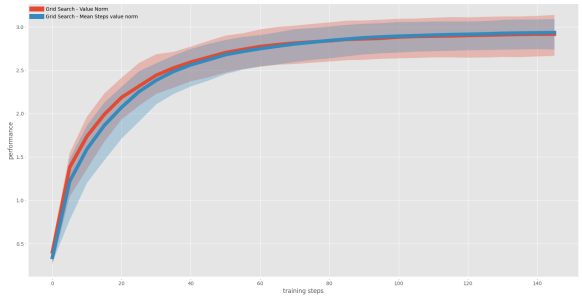

FIGURE 15 – Nombre de pas moyens par épisode : GridSearch (Max norm) vs (Min steps)

FIGURE 16 – Norme des valeurs moyennes par épisode : GridSearch (Max norm) vs (Min steps)

On constate clairement que si on maximise la norme ou qu'on minimise le nombre de pas moyens, nous n'avons aucune différence significative en ce qui concerne la norme du vecteur V. Cependant, on en constate quelques-unes lorsqu'il s'agit du nombre de pas moyens.

— **Bayesian Optimisation (Max norm) vs (Min steps)**

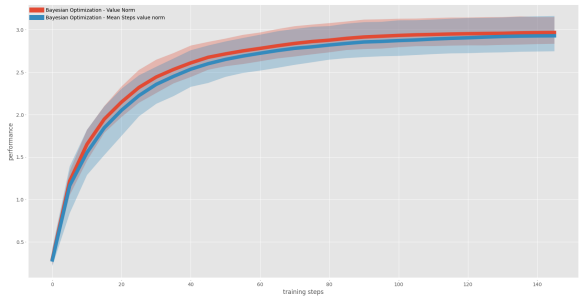

FIGURE 17 – Nombre de pas moyens par épisode : BayesianOptimisation (Max norm) vs (Min steps)

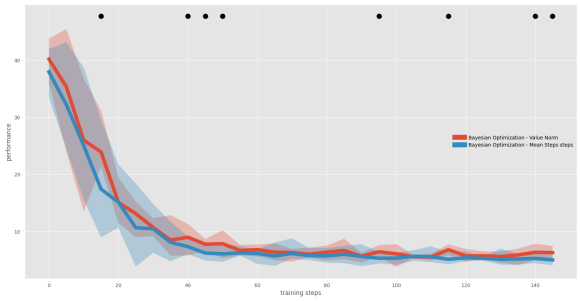

FIGURE 18 – Norme des valeurs moyennes par épisode : Bayesian Optimisation (Max norm) vs (Min steps)

De même avec l'optimisation bayésienne, on remarque qu'il n'y a aucune différence significative dans le graphe de la valeur de la norme, mais un peu plus dans le cas du nombre de pas moyens.

On peut en conclure que les deux types de métriques atteignent des résultats similaires, mais nous avons choisi le nombre de pas moyens car c'est une métrique bien plus interprétable dans notre cas.

V Conclusion

En conclusion, les nombreux tests statistiques montrent clairement que les valeurs trouvées par les algorithmes de recherche sont bien meilleures que les valeurs par défaut. Cependant, il est évident que ces résultats ne peuvent pas être généralisés à d'autres environnements avec des configurations différentes . Pour cela, il serait nécessaire d'investir davantage en puissance de calcul.

VI Perspectives

Pour surmonter les problèmes énoncés dans la conclusion, nous pourrions envisager d'entraîner les modèles sur un plus grand nombre d'environnements avec des configurations différentes, en faisant varier :

— La taille du labyrinthe

— Le nombre de labyrinthes

Cela impliquerait de constituer une base de données que nous subdiviserions en ensembles d'entraînement et de test. Nous pourrions ainsi rechercher les meilleurs hyperparamètres sur l'ensemble d'entraînement et les valider sur l'ensemble de test.

VII Annexe

Le code source complet du mini- projet, est disponible sur notre dépôt GitHub. Vous pouvez le consulter et le télécharger en suivant le lien ci-dessous :

— GitHub - Actor-Critic: Tuning Hyperparameters