FACULTÉ
DES SCIENCES
UNIVERSITÉ D'ANGERS

Engagement de non-plagiat

Je, soussigné(e), Touhami Chaïmaâ.....

déclare être pleinement conscient(e) que le plagiat de documents ou d'une partie d'un document publiés sur toutes formes de support, y compris l'internet, constitue une violation des droits d'auteur ainsi qu'une fraude caractérisée.

En conséquence, je m'engage à citer toutes les sources que j'ai utilisées pour écrire ce rapport ou mémoire.

Nom-Prénom : Touhami Chaïmaâ.....

Cet engagement de non-plagiat doit être inséré en première page de tous les rapports, dossiers, mémoires.

FACULTÉ DES SCIENCES
UNIVERSITÉ D'ANGERS

LERIA

Étude de l'émergence par neuro-évolution de nouveaux opérateurs de recherche locale pour résoudre des problèmes combinatoires

Chaïmaâ Touhami

MASTER 2
Informatique IA

Mars - Août 2024

Tuteur de stage :
Olivier Goudet

Encadrants :
Olivier Goudet
Frédéric Saubion
Adrien Goëffon

Table des matières

Remerciements

J'aimerais remercier mes encadrants qui m'ont aidée tout au long de mon stage.

This research used resources of the GLiCID Computing Facility (Ligerien Group for Intensive Distributed Computing, https://doi.org/10.60487/glicid, Pays de la Loire, France)

1 Introduction

Les algorithmes de recherche locale sont des algorithmes heuristiques permettant de trouver des solutions optimales pour des problèmes combinatoires difficiles. Ils commencent leur recherche à partir d'une solution initiale et passent d'une solution à une autre se trouvant dans un espace de recherche de voisins afin d'en trouver une qui soit meilleure que la précédente. Cependant, la recherche locale a pour faiblesse de pouvoir se retrouver coincée dans un optimal local. C'est pour cela que des algorithmes de recherche locale stochastique ont été introduits afin de pallier ce problème en donnant la possibilité de choisir une solution dégradante permettant de pouvoir sortir d'un optimal local. Les algorithmes évolutionnaires font partie des méthodes de recherche locale stochastiques qui ont fait leurs preuves pour optimiser des problèmes complexes en faisant évoluer une population de solutions. Cette population est construite à partir de solutions initiales et se verra évoluer en mutant, en combinant ou en sélectionnant certaines de ses solutions. Chaque solution est passée à une fonction d'évaluation afin de déterminer son score de *fitness*, c'est-à-dire à quel point celle-ci optimise le problème [1].

L'objectif de mon stage sous la direction d'Olivier Goudet au sein du LERIA [1] a été de démontrer si, à l'aide d'un algorithme évolutionnaire, nous pouvions optimiser les paramètres d'un réseau de neurones afin de pouvoir faire émerger de nouvelles stratégies d'optimisation de problème combinatoire. Pour cela, nous allons reprendre les travaux de O. Goudet, M. S. Amri Sakhri, A. Goëffon et F. Saubion où un algorithme neuro-évolutionnaire nommé Neuro-LS utilisant la méthode CMA-ES pour optimiser les paramètres d'un réseau de neurones a été développé dans le but maximiser la fonction objectif du problème combinatoire des paysages NK [2]. La stratégie apprise par ce dernier a pu ensuite être analysée et comparée à des méthodes de recherche locale déterministes. Celle-ci a réussi à atteindre de meilleurs scores de *fitness* en apprenant à effectuer une action (mutation) dégradante pour pouvoir sortir d'un maximum local. Nous pouvons reconnaitre le comportement d'un algorithme de recherche locale stochastique comme décrit précédemment. Ici, la seule information donnée au réseau de neurones et aux méthodes de recherche locale sont les différences de score de *fitness* à laquelle mènerait chaque action. Nous essayerons donc d'explorer d'autre manière d'améliorer cet algorithme neuro-évolutionnaire en fournissant plus d'information sur l'état de la recherche, en modifiant le traitement sur la sortie du réseau de neurones et en changeant la manière de faire évoluer les paramètre du réseau de neurones en remplaçant le CMA-ES par deux algorithmes d'apprentissage par renforcement différents. Ces modifications nous permettront de faire émerger plusieurs stratégies plus adaptées à l'état de la recherche.

1. Laboratoire d'Étude et de Recherche en Informatique d'Angers

2 Problème des paysages NK

Le problème des paysages NK (NK) est un problème combinatoire NP-complet caractérisé par un vecteur de bits de taille N où chaque bit est lié à K autres bits de ce même vecteur [3,4]. Nous avons donc un problème défini sur l'espace de recherche $\mathcal{S} := \{0,1\}^N$ de taille 2^N que nous pouvons imaginer sous la forme d'un paysage en N dimensions. Le voisinage d'un état $s \in \mathcal{S}$ sont les N états $s_{flip_i} \in \mathcal{S}$ résultant de la fonction de *bit flip* $flip(s,i)$, $\forall i \in [0..N[$. K lui est le paramètre déterminant la rugosité du paysage. Dans la Figure 1, nous pouvons voir que pour un K faible, ici $K = 1$, la surface du paysage est lisse avec une seule solution optimale rendant donc le problème strictement convexe. Cependant, plus K est grand, plus la surface est rugueuse, rendant le problème bien plus complexe à optimiser, car nous avons maintenant plusieurs maximums locaux.

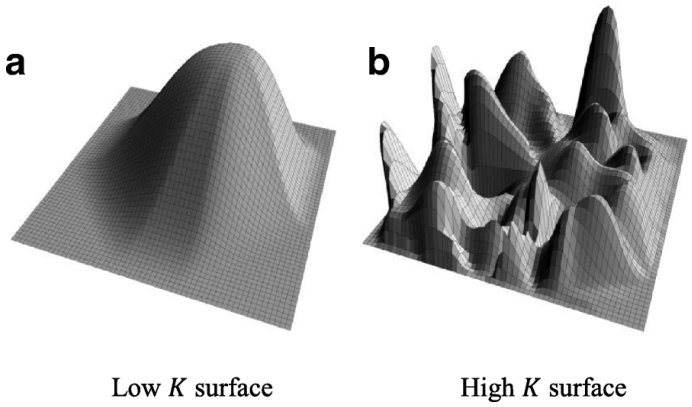

FIGURE 1 – Représentation de la rugosité de la surface du paysage en fonction de K [4]

La fonction de *fitness* F prenant en paramètre un état s (1) dépend de la *fitness-contribution* d'un bit s_i avec les K autres bits dont il dépend. Il est alors construit une matrice de contribution C de taille $2^N \times (K+1)$ définie sur $[0,1[$ où sont regroupés toutes les *fitness-contributions*. C'est lors de génération d'une instance du problème NK que cette matrice est remplie par des valeurs aléatoires de manière uniforme.

$$F(s) = \frac{1}{N} \sum_{i=0}^{N-1} C_i(s_i, D_i(s)) \tag{1}$$

Avec $D_i(s)$, le tuple des bits $s_j \in [1..K]$ auxquels s_i dépend.

Nous notons $o \in [0,1]^N$ le vecteur d'observations de différence de fitness entre un état $s \in \mathcal{S}$ et ses voisins :

$$\begin{aligned} o &= (o_0, ..., o_{N-1}) \\ &= (F(s) - F(flip(s,0)), ..., F(s) - F(flip(s,N-1)) \end{aligned} \tag{2}$$

Dans la suite de ce papier, quand nous parlerons d'effectuer une action $a \in \mathcal{A}$ sur un état $s \in \mathcal{S}$, cela sera l'équivalent d'effectuer la fonction $flip(s, a)$. Nous notons τ la trajectoire de taille T pour une instance du problème NK partant d'un état s_0 initial, le vecteur des couples $(s_t, a_{t+1}),\ \forall t \in [0, T[$.

3 Algorithmes étudiés

3.1 CMA-ES

Le CMA-ES [2] [5][6] est un algorithme évolutionnaire permettant de trouver des solutions pour des problèmes d'optimisation dits boîte-noire.

Le principe du CMA-ES est d'échantillonner une population de solutions X suivant une loi normale multidimensionnelle (3) où n est la taille du problème, $X \in \mathbb{R}^n$ une solution, $m \in \mathbb{R}^n$ le vecteur de moyenne, $\sigma \in \mathbb{R}^+$ l'écart type et $C \in \mathbb{R}^{n \times n}$ la matrice de covariance :

$$X \sim \mathcal{N}(m, \sigma^2 C) \tag{3}$$

Ensuite, chaque solution se verra passer à la fonction d'évaluation afin de leur attribuer un score de *fitness* et de pouvoir mettre à jour les paramètres pour le prochain échantillonnage de solutions. Ces étapes sont répétées pendant G générations.

Comme on peut le voir dans la Figure 2, nous avons plusieurs solutions (points noirs) qui sont tirés en fonction de la distribution de la population (tirets orange). Le centre de la distribution est déterminé par m et la largeur de l'ellipse par C. À chaque génération, nous pouvons remarquer l'ellipse se déplace et se déforme sur le paysage afin de trouver la zone optimale et réduire dessus pour pouvoir générer des solutions avec un haut score de fitness.

Les paramètres d'échantillonnage

— **La moyenne m** initialisée par une solution $Y \in \mathbb{R}^n$ permet de positionner le centre de la distribution de la population dans l'espace de recherche. La valeur de $m^{(0)}$ est importante, car celle-ci va influer sur le nombre de générations qu'il va falloir pour trouver la zone d'optimalité.

À chaque génération, nous calculons la nouvelle moyenne $m^{(g+1)}$ en fonction de la moyenne précédente $m^{(g)}$de la vitesse d'apprentissage α et de la moyenne pondérée des μ meilleures solutions x triées dans l'ordre décroissant de leur score de *fitness* parmi les λ solutions échantillonnées dy :

$$m^{(g+1)} = m^{(g)} + \alpha \cdot dy \tag{4}$$

2. *Covariance Matrix Adaptation - Evolution Strategy*

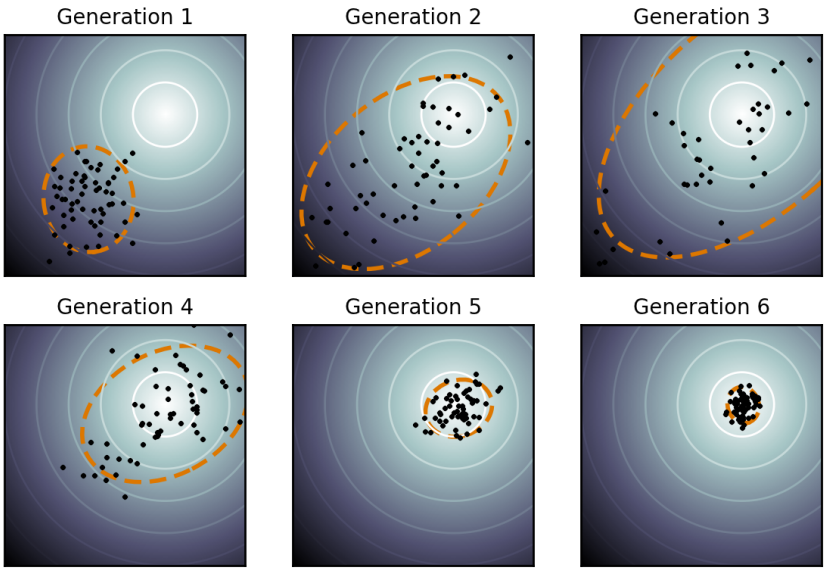

FIGURE 2 – Optimisation de solutions pour un problème en deux dimensions suivant l'évolution de la matrice de covariance

$$dy = \sum_{i=1}^{\mu} w_i x_{i:\lambda}^{(g+1)} \tag{5}$$

$$\sum_{i=1}^{\mu} w_i = 1, \ w_1 \geq w_2 \geq ... \geq w_\mu \geq 0 \tag{6}$$

— **La trajectoire d'évolution $\mathbf{p_c} \in \mathbb{R}^n$** initialisée à 0^n est la somme des étapes de recherche permettant de l'information de signe dans le calcul de $C^{(g+1)}$. Le calcul de $p_c^{(g+1)}$ peut être fait en faisant un lissage exponentiel :

$$p_c^{(g+1)} = (1 - c_c)p_c^{(g)} + h_\sigma^{(g+1)}\sqrt{c_c(2 - c_c)\mu_w}dy \tag{7}$$

Avec c_c le facteur d'accumulation.

$$\mu_w = \frac{1}{\sum_{i=1}^{\mu} w_i^2} \tag{8}$$

Avec $h_\sigma^{(g+1)}$ la fonction d'Heaviside :

$$h_\sigma^{(g+1)} = \begin{cases} 1 & \text{si } \frac{||p_\sigma^{(g+1)}||}{\sqrt{1-(1-c_\sigma^{2(g+2)}\mathbb{E}[||\mathcal{N}(0,I)||]}} < 1.4 + \frac{2}{n+1}, \\ 0 & \text{sinon} \end{cases} \tag{9}$$

$$\mathbb{E}[||\mathcal{N}(0, I)||] \approx \sqrt{d}(1 - \frac{1}{4n} + \frac{1}{21n^2}) \tag{10}$$

— **La trajectoire d'évolution conjuguée $\mathbf{p}_\sigma$** est assez proche dans sa définition à celle de p_c, cependant, celle-ci va aussi dépendre de la matrice de covariance C (11). p_σ permettra de mettre à jour la valeur de σ.

$$p_\sigma^{(g+1)} = (1 - c_\sigma)p_\sigma^{(g)} + \sqrt{c_\sigma(2 - c_\sigma)\mu_w}\sqrt{C^{(g)}}^{-1}dy \tag{11}$$

Avec c_σ le facteur d'accumulation.

Nous pouvons voir dans la Figure 3 que pour trois scénarios avec le même nombre d'étapes de même taille, nous avons trois tailles de trajectoires d'évolutions, car ces dernières vont dépendre de la direction de ces étapes. De ce fait, la taille de la trajectoire d'évolution va permettre de réduire la valeur de σ quand celle-ci est petite et l'augmenter quand elle est grande. Ainsi, l'évolution vers la zone optimale peut se faire plus rapidement quand nous savons que nous allons dans la bonne direction ou ralentir pour éviter de la rater.

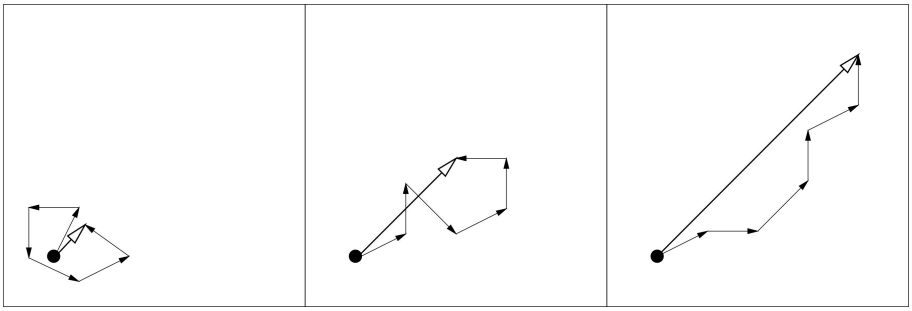

FIGURE 3 – Trois trajectoires d'évolution différentes dans la direction que prend chaque étape [5]

— **La taille de pas (et écart-type)** σ initialisée par une valeur $s^{(0)} > 0$ est la variable qui va multiplier la matrice de covariance C et ainsi permettre d'accélérer la recherche comme dit précédemment.

$$\sigma^{(g+1)} = \sigma^{(g)} \exp\left(\frac{c_\sigma}{d_\sigma}\left(\frac{||p_\sigma^{(g+1)}||}{\mathbb{E}[||\mathcal{N}(0, I)||]} - 1\right)\right) \tag{12}$$

Avec $\mathbb{E}[||\mathcal{N}(0, I)||]$ la taille attendue de $||p_\sigma^{(g+1)}||$.

— **La matrice de covariance C** initialisée par la matrice d'identité I de taille $n \times n$ est le paramètre qui va définir la forme de la distribution de la population dans l'espace de recherche, comme nous pouvons le voir dans la Figure [4].

$$\begin{bmatrix} Var(x_0) & ... & Cov(x_{n-1}, x_0) \\ ... & ... & ... \\ Cov(x_{n-1}, x_0) & ... & ...Var(x_{n-1}) \end{bmatrix} \tag{13}$$

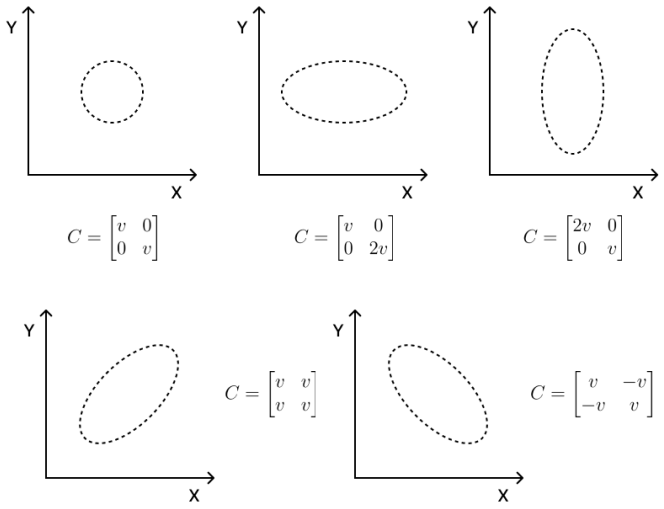

FIGURE 4 – Différentes formes que peut prendre la distribution de la population (tirets noirs) en fonction de la valeur de la matrice de covariance C pour un problème de taille $n = 2$

Celle-ci est mise-à jour comme suit :

$$\begin{aligned} C^{(g+1)} = &\big(1 + (1 - h_\sigma^{(g+1)})c_1 c_2 (2 - c_c)\big) C^{(g)} \\ &+ c_1 \left[p_c^{(g+1)} (p_c^{(g+1)})^T - C^{(g)} \right] \\ &+ c_\mu \sum_{i=1}^{\lambda} w_i' \left[x_{i:\lambda} y_{i:\lambda}^T - C^{(g)} \right] \end{aligned} \tag{14}$$

Avec c_1 et c_μ des vitesses d'apprentissage.

$$w_i' := w_i \cdot \begin{cases} 1 & \text{si } w_i \geq 0 \\ \frac{n}{||\sqrt{C^{(g)}}^{-1} x_{i:\lambda}||^2} & \text{sinon} \end{cases} \tag{15}$$

Algorithme Neuro-LS

Comme dit dans l'introduction, nous allons reprendre l'algorithme nommé Neuro-LS de [2]. Celui-ci a pour objectif d'optimiser les paramètres θ d'un réseau de neurones au moyen du CMA-ES. De ce fait, l'espace de recherche pour le CMA-ES est ici l'espace de paramètres Θ, car nous voulons trouver les paramètres $\theta \in \Theta$ optimaux pour le réseau de neurones. Une solution θ est évaluée en fonction du score de *fitness* moyen obtenue par la politique π_θ apprise sur un *batch* de trajectoires τ. À partir de ces évaluations, les paramètres d'échantillonnage sont mis à jour, et la solution s obtenant la plus haute évaluation est retenue pour mettre à jour ceux du réseau de neurones.

Algorithm 1: Deep CMA-ES

Data: G, s_0, σ
π_θ = init_policy();
es = cmaes.init(s_0, σ);
P_{size} = es.popsize();
$r_{best} = -\infty$;
$P_{fitness}$ = [];
for *g in 0..G* **do**
  P = es.ask();
  for $x \in P$ **do**
    r_{max} = get_trajectory(π_θ, s_0);
    $P_{fitness}$.append(r_{max});
    if $r_{max} > r_{best}$ **then**
      $r_{best} = r_{max}$;
      $x_{best} = x$;
    end
  end
  es.tell(P, $P_{fitness}$);
  π_θ.update_weights(x_{best})
end

3.2 Apprentissage par Renforcement

L'apprentissage par renforcement (AR) est une discipline de l'apprentissage automatique. Le concept est, comme on peut le voir dans la Figure 5, d'entraîner un agent à apprendre une politique π en lui faisant faire des actions $a \in A$ qui vont modifier son environnement. Il lui sera ensuite retourné le nouvel état $s \in S$ de son environnement ainsi qu'une récompense r qui représente l'impacte positif ou négatif de son action.

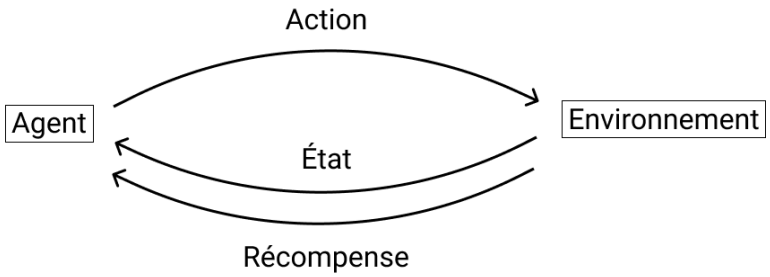

FIGURE 5 – Concept d'apprentissage par renforcement d'un agent en récompensant ses actions

Comme, ici, nous allons allier un algorithme d'apprentissage par renforcement à un réseau de neurones, nous pouvons alors parler d'Apprentissage par Renforcement

Profond (ARP).

3.2.1 REINFORCE

Le premier algorithme d'AR que nous allons étudier est celui du REINFORCE [7]. Celui-ci est *policy-gradient* et va donc apprendre une politique π en la mettant à jour en fonction de sa récompense $R(\tau)$ sur une trajectoire τ que nous chercherons à maximiser :

$$\nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta}[R(\tau)] = \mathbb{E}[\nabla_\theta \log \pi_\theta(\tau)(R(\tau) - b)] \tag{16}$$

Comme le REINFORCE a pour faiblesse d'avoir une haute variance, nous introduisons b la *baseline* permettant de la réduire [8].

$$\nabla_\theta \log \pi_\theta(\tau) = \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_{t+1}|s_t) \tag{17}$$

$$\pi_\theta(\tau) = s_0 \prod_{t=0}^{T} \pi_\theta(a_{t+1}|s_n) S(a_{t+1}, s_n) \tag{18}$$

Algorithme

Pour calculer la récompense $R(\tau)$, nous retenons le score de *fitness* maximal r_{max} parmi toutes les récompenses r_t calculées lors de la trajectoire. Ainsi, c'est la suite de toutes actions a_t effectuées sur un état initial s_0 qui est récompensée. La *baseline* b se mettra à jour à chaque génération en fonction du score de *fitness* de validation. Nous avons aussi le paramètre α qui est la vitesse d'apprentissage qui sera donné à l'optimiseur *Adam*.

Algorithm 2: REINFORCE

Data: G, s_0, α
π_θ = init_policy();
opti = optim.Adam(θ, α, maximize=True);
b = 0.5;
for *g in 0..G* **do**
  $\nabla_\theta \log \pi_\theta(\tau)$, $R(\tau)$ = get_trajectory(π_θ, s_0);
  loss = $\nabla_\theta \log \pi_\theta(\tau) \times (R(\tau) - \text{b})$;
  opti.step();
  b = get_validation_score(π_θ);
end

3.2.2 PPO avec pénalité KL adaptative

Le PPO[3] [9] est une méthode d'ARP qui, au lieu de seulement apprendre une politique π_τ en fonction du résultat final d'une trajectoire, va essayer de maximiser l'objectif en comparant deux politiques pendant E *epoch* en fonction de couples $(s_t, a_t)_t$ pour des $t \in [0, T[$ choisis aléatoirement et de la récompense $R(\tau)$.

— $\pi_\theta(a_t|s_t)$ la politique apprise à la génération $g-1$ et à l'*epoch* e_{final}

— $\pi'_\theta(a_t|s_t)$ la politique apprise à la génération g et à l'*epoch* $e \in [0, E[$

Cependant, pour éviter que l'écart entre deux politiques π_θ et π'_θ soit trop large et d'ainsi faire des sauts trop grands dans l'apprentissage, nous allons aussi introduire le concept de pénalité KL[4] adaptative (KL). Ce dernier permet de pénaliser l'apprentissage si l'écart entre π_θ et π'_θ est trop éloigné de la divergence δ souhaitée. Nous avons alors maintenant la formule de maximisation pour le PPO-KL comme suit :

$$\max_\theta \quad \mathbb{E}_t\left[\frac{\pi'_\theta(a_t|s_t)}{\pi_\theta(a_t|s_t)}(R(\tau)-b) - \beta \mathrm{D}_{\mathrm{KL}}(\pi_\theta(\cdot|s_t), \pi'_\theta(\cdot|s_t))\right] \tag{19}$$

Avec D_{KL} la fonction calculant la divergence KL en fonction de deux distributions de probabilités discrètes P et Q :

$$\mathrm{D}_{\mathrm{KL}}(P, Q) = \sum_{x \in \mathcal{X}} P(x) \log\left(\frac{P(x)}{Q(x)}\right) \tag{20}$$

Et β le coefficient de pénalité qui se met à jour à chaque fin de génération où $\delta_{\mathrm{KL}} = \mathbb{E}\left[\mathrm{D}_{\mathrm{KL}}(\pi_\theta(\cdot|s_t), \pi'_\theta(\cdot|s_t))\right]$:

$$\beta = \begin{cases} \frac{\beta}{2} & \text{si } \delta_{\mathrm{KL}} < \frac{\delta}{1.5} \\ 2\beta & \text{si } \delta_{\mathrm{KL}} > 1.5\delta \\ \beta & \text{sinon} \end{cases} \tag{21}$$

Algorithme

Comme pour le REINFORCE, $R(\tau)$ sera égale au score de *fitness* maximal rencontré au cours de la trajectoire τ. Le paramètre α et la variable b sont aussi ici les mêmes que pour le REINFORCE.

3. Proximal Policy Optimization
4. Kullback-Leibler

Algorithm 3: PPO-KL

Data: $G, E, s_0, \alpha, \beta, \delta$
π_θ = init_policy();
target = 0.5;
for *g in 0..G* **do**
  $A_\tau, S_\tau, p_t, r_{max}$ = get_trajectory(π_θ, s_0);
  opti = optim.Adam(θ, α, maximize=True);
  for *e in 0..E* **do**
    a_t, s_t = random_t(A_τ, S_τ);
    $p'_t = \pi_\theta(a_t, s_t)$;
    δ_{KL} = kl_divergence(p_t, p'_t);
    loss = $E_t\left[\frac{p'_t}{p_t}(r_{max} - \text{target}) - \beta\delta_{\text{KL}}\right]$;
    opti.step();
  end
  if $\delta_{\text{KL}} < \frac{\delta}{1.5}$ **then**
    $\beta = \frac{\beta}{2}$;
  end
  if $\delta_{\text{KL}} > 1.5\delta$ **then**
    $\beta = 2\beta$;
  end
  target = get_validation_score(π_θ);
end

4 Structure du réseau de neurones

La structure du réseau de neurones que nous entraînerons sera la même que celle de [2]. Dans la Figure 6, nous avons à l'entrée du réseau de neurones un vecteur d'observation $o \in \mathbb{R}^N$ qui correspond à l'écart de fitness observé pour chaque action $a \in A_s$ faite sur un état $s \in \mathcal{X}$. Ce vecteur d'observation o va produire une moyenne de ses valeurs $\rho(o) \in \mathbb{R}$ (23) qui sera incorporée à la prochaine couche. Le même processus sera fait à la sortie de chaque couche cachée ϕ_{θ_j} (23). La dernière couche donne un vecteur $g_\theta(o) \in \mathbb{R}^N$ auquel sera appliquée la fonction $Argmax$ afin de déterminer quelle action a sera la plus adaptée à effectuer selon la politique $\pi_\theta(o)$ apprise. Chaque couche a une taille d'entrée l_j et de sortie l_{j+1}, sachant que $l_0 = 1$ et $l_{P_1} = 1$ où P est le nombre de couches cachées.

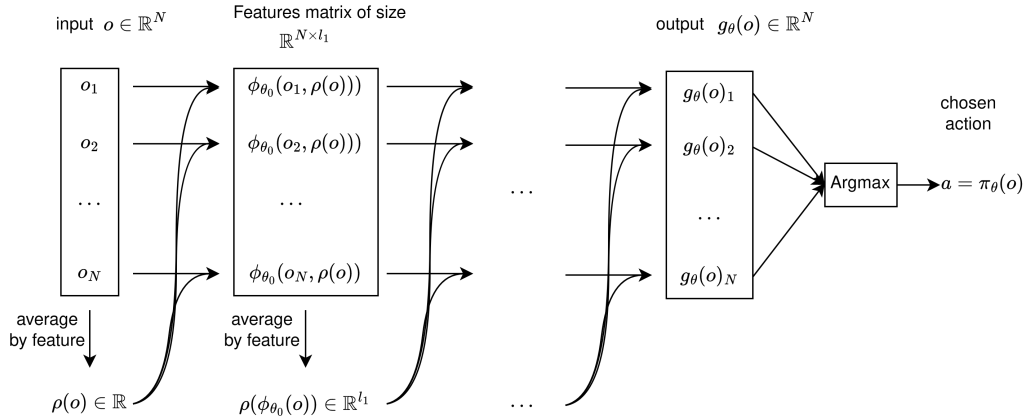

FIGURE 6 – [2]

$$\rho(v) = \frac{1}{N} \sum_{i=1}^{N} v_i \tag{22}$$

Où v est un vecteur de nombres réels.

$$\phi_{\theta_j}(v)_i = \eta(\beta_j + x_i W_j + \rho(v)\Gamma_j) \tag{23}$$

Avec $\eta(x)$ la fonction d'activation Tanh définie sur $\eta : \mathbb{R}^{l_j} \to \mathbb{R}^{l_{j+1}}$ par :

$$\eta(z) := (\tanh(z_1, ..., \tanh(z_{l_j})) \tag{24}$$

Les paramètres $\theta := \{(W_j, \Gamma_j, \beta_j)\}_{j \in [\![0,P]\!]}$ de ce réseau de neurones sont composés de la matrice de poids $W_j \in \mathbb{R}^{l_j \times l_{j+1}}$ pour le traitement individuel de chaque valeur, la matrice de poids de moyenne $\Gamma_j \in \mathbb{R}^{l_j \times l_{j+1}}$ qui traite la moyenne de toutes les valeurs et enfin le vecteur de biais $\beta_j \in \mathbb{R}^{l_j}$.

Nous étudierons aussi un autre réseau de neurones ne prenant pas en compte la moyenne $\rho(v)$ et n'aura donc pas la matrice de poids Γ dans ces paramètres θ. Cela

permettra de voir l'impact de la moyenne sur la politique apprise. Nous appellerons le réseau de neurones avec moyenne *remix* (R) et celui sans moyenne *no remix* (NR).

5 Méthodes étudiées

Nous allons avoir plusieurs méthodes de recherche locale que nous comparerons à des méthodes de Neuro-LS. Ces dernières seront différentes sur comment elles traitent les informations en entrée et en sortie du réseau de neurones. Nous étudierons alors les différentes stratégies émergeant des politiques apprises par les différentes méthodes Neuro-LS.

5.1 Recherche locale

Il a été fait en sorte que toutes les méthodes de recherche locale que nous allons voir soient déterministes en tirant tous nombres pseudo-aléatoires en fonction d'une graine h. Cette dernière est obtenue au moyen d'une fonction de hachage prenant en paramètre l'état s_t [2].

— ***Best improvement Hill Climbing*** (BHC^+) **:** Méthode faisant toujours l'action a_i correspondant à la meilleure observation $o_i > 0$. Si $\forall o_i \in o\, o_i < 0$, alors l'action à effectuer est tirée de manière aléatoire. [Cette méthode est optimale pour résoudre les problèmes convexes [10]].

— ***First improvement Hill Climbing*** (FHC^+) **:** Méthode tirant aléatoirement des actions a_i et sélectionnant la première ayant une observation $o_i > 0$. Si $\forall o_i \in o\, o_i < 0$, alors l'action à effectuer est tirée de manière aléatoire.

— $(\mathbf{1}, \lambda)$***-Evolution Strategy*** $(1, \lambda) - ES$ **:** Méthode tirant aléatoirement λ actions et sélectionnant celle ayant l'observation la plus haute, peu importe le signe de cette dernière. La recherche pour chaque $\lambda \in [\![1, N-1]\!]$ est effectuée et celui donnant le meilleur score de *fitness* en entrainement est retenue [11].

— ***Tabu*** **:** Méthode sélectionnant l'action a_i avec la plus haute observation o_i parmi les actions non taboues. Une action taboue est une action qui est interdite d'effectuer, car celle-ci a déjà été choisie récemment. Pour cela, la stratégie garde en mémoire quand pour la dernière fois une action a été sélectionnée sous la forme d'un vecteur $tabu \in R^{+\,N}$ initialisé à 0^N et mis à jour à la fin de chaque tour comme suit :

$$tabu_{a_t} = \left\lfloor \frac{N}{10} \times random() \right\rfloor + \alpha + t \tag{25}$$

Avec $random()$ la fonction donnant un nombre aléatoire rand $\in [0, 1[$ et $\alpha \in [\![1, N-1]\!]$ un paramètre de pénalité (plus α est grand, plus il faut de

tours avant qu'une action ne soit plus taboue).

5.2 Neuro-LS

— ***NN*** **:** Méthode apprise comme défini dans la Section 4 avec en entrée le vecteur d'observation o et en sortie le vecteur $g_\theta(o)$ auquel est appliquée la fonction $argmax$. C'est la stratégie qui a été étudiée dans [2] et à laquelle nous appliquerons plusieurs modifications.

— ***NNSoftmax*** **:** Méthode affectant la fonction $softmax$ sur la sortie g_θ du réseau de neurones à la place de la fonction $argmax$ pour ensuite échantillonner l'action a à faire sur l'état s selon la distribution de probabilités discrètes obtenue.

— ***NNTabu*** **:** Méthode où nous avons en entrée le vecteur d'observations o ainsi qu'un vecteur d'indication d'utilisation des actions $\text{ind} \in [0,1]^N$ initialisé à $\{0\}^N$ au début d'une trajectoire est mis à jour comme suit à la fin de chaque fin de tour t :

$$\text{ind}_{t+1} = \coprod_{i=0}^{N} \begin{cases} 1 - \frac{\min(t+1-\text{turns}_{t+1,i},N)}{N} & \text{si turns}_i > 0 \\ \text{ind}_{t,i} & \text{sinon} \end{cases} \quad [5] \tag{26}$$

Avec $\text{turns} \in [\![0,T]\!]^N$ le vecteur tenant compte à quel tour les actions ont été choisies pour la dernière fois initialisé à $\{0\}^N$ qui se met à jour juste avant ind en changeant la valeur turns_a par $t+1$.

— ***NNTabuSoftmax*** **:** Méthode où on a la même matrice en entrée que pour *NNTabu* et la même application sur la sortie que pour *NNSoftmax*.

— ***NNRanked*** **:** Méthode où les valeurs du vecteur d'observations o sont d'abords triées pour être séparées en fonction de leur signe afin de pouvoir construire le vecteur $\text{rank} = [|o_-| - 1..0..|o_+|]$, avec $|o_-|$ et $|o_+|$ le nombre de valeurs négatives et positives respectivement dans o. Ensuite, rank est normalisé dans l'interval $[-1,1]$ puis retrié en fonction de l'ordre de o avant d'être passé en entrée du réseau de neurones.

— ***NNReinforce, NNPPOKL*** **:** Méthodes équivalentes à *NNSoftmax* adaptées pour le REINFORCE et le PPO-KL respectivement.

— ***NNReinforceTabu, NNPPOKLTabu*** **:** Méthodes équivalentes à *NNTabuSoftmax* adaptées pour le REINFORCE et le PPO-KL respectivement.

5. $\coprod$ la fonction de concaténation

6 Expériences

Grâce aux expériences qui vont suivre, nous allons voir si utiliser l'optimisation par neuro-évolution peut nous permettre d'obtenir de meilleurs résultats que les méthodes de recherche locale. Nous observerons aussi le comportement des stratégies apprises pour comprendre leur raisonnement pour choisir une action a en prenant en compte la moyenne du vecteur d'observation $\rho(o)$ ou non.

6.1 Configurations

Les expériences ont été faites en Python 3.12.2 sur les GPU NVIDIA et les CPU Intel du cluster Waves du CCIPL (GLiCID) à l'aide du *framework* PyTorch 2.3.0 CUDA 12.1. Pour le CMA-ES, la librairie *cmaes* [6] a été utilisé. Nous travaillerons avec des instances du problème NK $N \in \{32, 64\}$ et $K = 8$. 1000 instances d'entraînement, 100 de validation et 100 de test ont été générées et utilisées de la même manière pour tous les modèles. Tous les tests ont été faits sur 10 *restarts*. À chaque génération, les modèles sont entraînés sur un *batch* d'instances de taille $B = 100$ choisi aléatoirement avec remise sur une trajectoire de taille $T = 2N$. Pour le CMA-ES, les modèles ont été entraînés sur 1000 générations avec un échantillonnage de 19 solutions par génération (P_X la taille de la population choisie automatiquement par la librairie). Or, l'appel à la fonction objectif est réalisé en tout $G \times B \times P_X$. De ce fait, les modèles pour le REINFORCE et le PPO-KL sont, eux, entraînés sur 19 000 générations, car l'appel à la fonction objectif est effectué $G \times B$ fois. Pour le CMA-ES, la taille de pas σ est initialisée à 0.2. Pour le REINFORCE et le PPO-KL, l'optimiseur Adam a été utilisé en mode maximisation et avec une vitesse d'apprentissage $\alpha = 0.005$. Pour le PPO-KL, nous aurons un nombre d'*epoch* $E = 10$, la divergence $\delta = 0.0001$ et la variable $\beta = 0.5$. Le réseau de neurones sera constitué de deux couches cachées de 10 et 5 neurones.

6.2 Les méthodes de recherche locale et la méthode *NN*

Ici, nous comparerons les méthodes de recherche locale BHC^+, FHC^+, $(1, \lambda) - ES$ et $Tabu$ à la méthode neuro-évolutionnaire NN pour observer si la méthode CMA-ES la plus simple dans les informations données au réseau de neurones ainsi que de sa manière d'en traiter sa sortie peu apprendre une stratégie plus efficiente pour maximiser la fonction objectif de notre problème.

Dans la Table 1, on peut remarquer que la méthode NN en R et NR performe mieux que les méthodes BHC^+, FHC^+ et $(1, \lambda) - ES$ et ce, en travaillant uniquement avec le vecteur d'observation o pour faire un choix sur l'action a à exécuter.

Méthodes	$N = 32$	$N = 64$
BHC^+	0.704	0.709
FHC^+	0.712	0.719
$(1, \lambda) - ES$	0.703	0.706
$Tabu$	0.755	0.759
$NN\ R$	0.733	0.737
$NN\ NR$	0.731	0.736

TABLE 1 – Résultats des tests pour les méthodes de recherche locale et de la méthodeNN avec un réseau de neurones R ou NR pour le problème NK-Landscape où $K = 8$

Pour pouvoir comprendre pourquoi, nous pouvons regarder la Figure 7 pour analyser la stratégie apprise afin de sélectionner une action a en fonction du vecteur d'observations o passé en entrée du réseau de neurones. Nous pouvons remarquer facilement la forte pente à 0 vers le pic à 0.5 pour les $\rho(o) < 0$. Ici, le choix est de privilégier les actions permettant de légèrement améliorer le score de *fitness* quand en moyenne ce dernier est négatif. Le fait de choisir une trop bonne action dans ce cas de figure pourrait coincer la recherche dans un maximum local. Il serait donc préférable de monter doucement pour ne pas rater d'autres opportunités d'améliorer le score de *fitness*. Cependant, quand $\rho(o) > 0$, comme pour $g(o)_0$, le pic à 0.5 est effacé pour laisser une courbe de moins en moins décroissante et même croissante pour un $\rho(o) > 0.5$ (cf. Figure 26 de vecteurs o générés pour mieux observer chaque cas de figure).

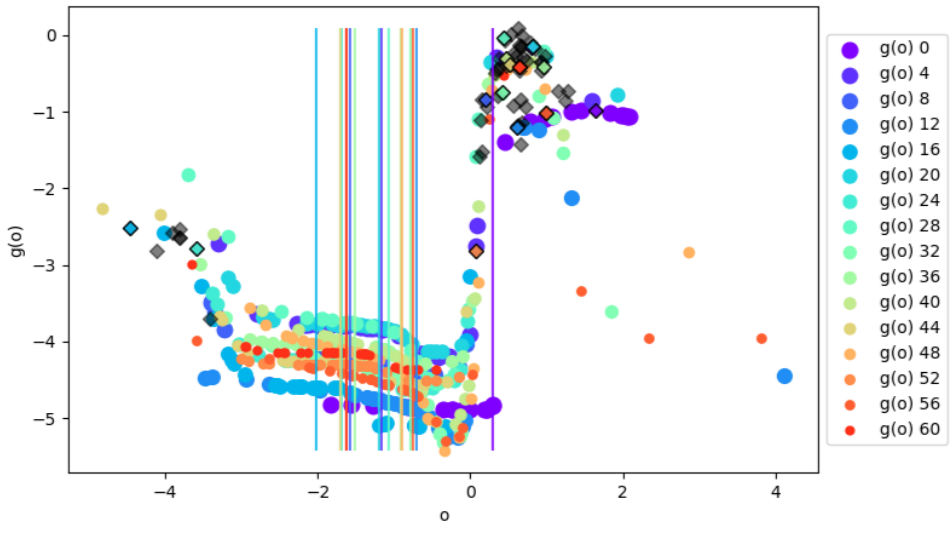

FIGURE 7 – Sorties $g_\theta(o)_t$ en fonction des vecteurs d'observations $o_{\cdot,t}$ sur une trajectoire de test pour la méthode NN R avec les $g_\theta(o_a)_t$ contourées correspondant aux actions a_t sélectionnées et les barres verticales aux moyennes $\rho(o_{\cdot,t})$ pour $N = 32$ et $K = 8$

Ainsi, ici, il vaut mieux faire un grand saut positif quand on sait qu'il y a aussi assez d'autres actions qui permettent aussi d'améliorer significativement le score de *fitness*. Enfin, nous pouvons aussi voir qu'un $g_\theta(o)_i$ est mieux valorisé si o_i est très négatif. De ce fait, comme un $argmax$ est appliqué à la sortie du réseau de neurones, alors l'action menant la plus grande baisse du score de *fitness* est sélectionnée si tous les o_i sont négatifs (ici $g_\theta(o)_{0,20}$ et $g_\theta(o)_{0,24}$). Nous pouvons comprendre que la stratégie est de faire une grande chute quand nous sommes coincés dans un maximum local afin de pouvoir découvrir d'autres moyens de remonter.

Par contre, cette fois-ci sur la Figure 8, quand le réseau de neurones est NR alors toutes les $g_\theta(o)$ se retrouvent sur la même courbe. Nous y retrouvons tout de même le pique à 0.5 privilégiant les faibles gains en score de *fitness* et nous pouvons aussi reconnaître la stratégie de choisir la pire action avec la pente légèrement croissante de 0 vers -6.

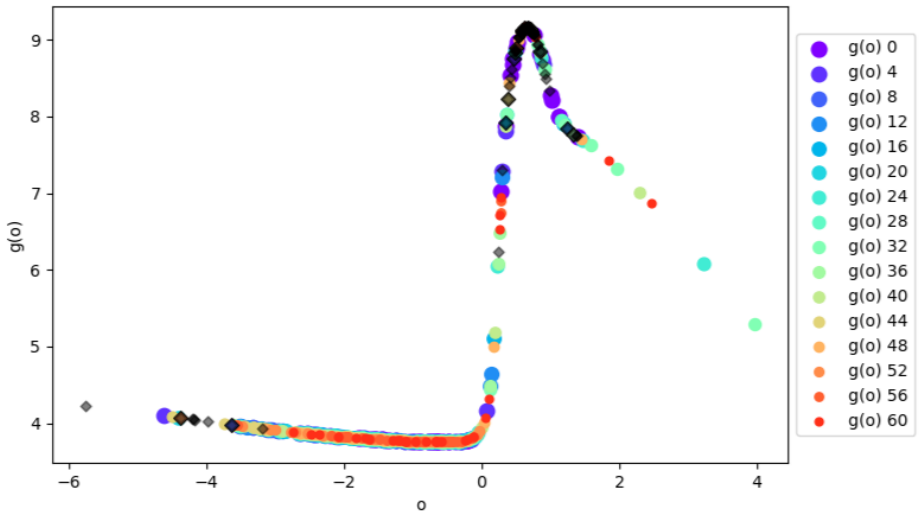

FIGURE 8 – Sorties $g_\theta(o)_t$ en fonction des vecteurs d'observations $o_{\cdot,t}$ sur une trajectoire de test pour la méthode *NN* NR avec les $g_\theta(o_a)_t$ contourées correspondant aux actions a_t sélectionnées s $\rho(o_{\cdot,t})$ pour $N = 32$ et $K = 8$

La similitude dans la prise de décision entre R et NR explique la faible différence de score de *fitness* en test. Cela peut aussi se remarquer dans la Figure 9 où l'évolution du score de *fitness* entre R et NR est semblable avec un léger avantage en moyenne pour R.

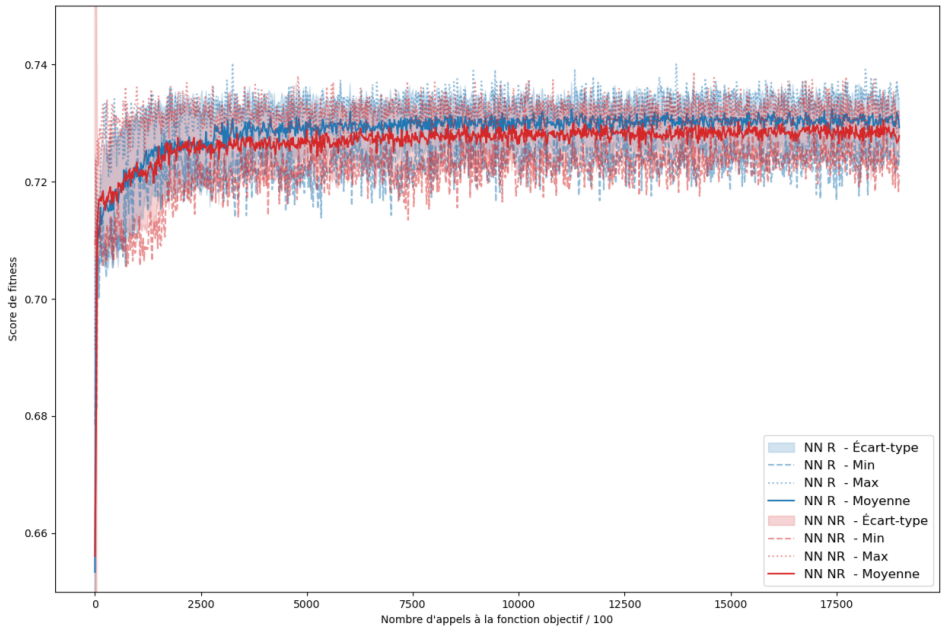

FIGURE 9 – Évolution du score de *fitness* en validation pour la méthode *NN* en N et NR en fonction du nombre d'appels à la fonction objectif / 100

Si nous revenons à la Table 1, la méthode de recherche locale *Tabu* reste tout de même la plus performante avec plus de 0.02 de score de *fitness* que la méthode *NN*. Cela s'explique par le fait que la méthode *Tabu* dispose d'une information en plus qui la liste des actions a interdites de choisir pour un certain de temps. Effectivement, le point faible de la méthode NN est que parfois la recherche se retrouve coincée dans un maximum local, car les mêmes actions sont effectuées en boucle (cf. Figure 27 où l'on alterne la pire action avec des actions légèrement améliorantes).

6.3 De la méthode *NN* à la méthode *NNTabu*

Pour que notre méthode neuro-évolutionnaire *NN* puisse rivaliser à la méthode de recherche locale *Tabu*, il faut que celle-ci puisse aussi tenir compte des actions effectuées dans le passé d'une trajectoire. C'est pour cela que nous allons maintenant nous intéresser à la méthode *NNTabu* retenant depuis combien de tours des actions ont été choisis. À la différence de la méthode *Tabu*, aucune interdiction n'est faite, mais seulement l'information sur les actions passées est fournie au réseau de neurones. Si nous regardons la Figure 10, nous avons les + et les losanges qui correspondent aux actions non taboues et les ronds aux actions taboues. La taille des ronds dépend de la valeur de l'action dans le vecteur tabu. Nous pouvons en premier lieu observer que nous avons deux courbes principales : une qui est tracée par les actions non taboues et une par les actions récemment taboue (tabu_a très proche 1^-). Nous retrouvons pour la première la forte croissance partant 0, seulement, cette fois-ci, nous avons deux pic séparés par une très légère décroissance avec un centré à 0.25 et

un à 1. Le second pic est suivi par une décroissance jusqu'à $o = 2$ pour ensuite être monotone. De plus, les observations négatives à partir de -2 à $+\infty$ sont maintenant très défavorisées. Nous pouvons alors comprendre qu'il est préférable de faire une action améliorant légèrement le score de *fitness* plutôt que de monter rapidement quand les actions ne sont pas taboues. Pour la seconde courbe, ce qui est intéressant de relevé est la croissance entre 0.5 et 0.75 suivie par une monotonie. Les valeurs pour une action taboue restent toujours inférieures à une action non taboue. De ce fait, l'information sur les actions choisies dans le passé d'une trajectoire a bien servi pour les défavoriser. Cependant, une action fortement taboue peut tout de même être choisie dans le cas où il n'y aurait pas d'actions moins ou pas taboue dans les zones très favorables ($[0, +\infty]$) ou que celle-ci correspond à la plus grande valeur de o.

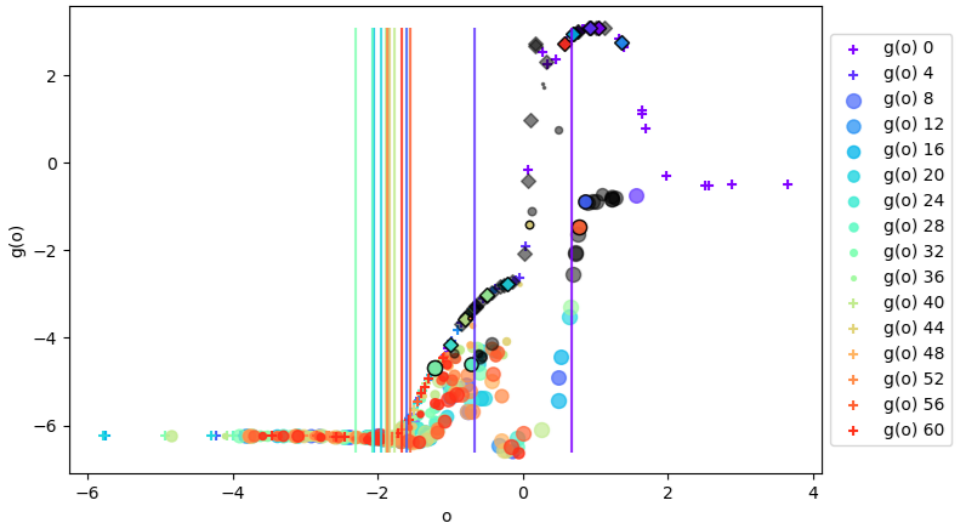

FIGURE 10 – Sorties $g_\theta(o)_t$ en fonction des vecteurs d'observations $o_{\cdot,t}$ sur une trajectoire de test pour la méthode *NNTabu* R avec les $g_\theta(o_a)_t$ contourées correspondant aux actions a_t sélectionnées et les barres verticales aux moyennes $\rho(o_{\cdot,t})$ pour $N = 32$ et $K = 8$

La stratégie de la méthode *NNTabu* en NR est semblable dans le sens où la courbe des actions non taboues est aussi monotone en $[-\infty, 0]$ et croissante en $[0, 2]$. La différence que nous pouvons noter est le fait qu'en $[0, 1]$, des actions moyennement à peu taboues sont mieux valorisées que des actions non taboues. Cela permettrait de réeffectuer une action légèrement à moyennement améliorante, même si celle-ci est taboue.

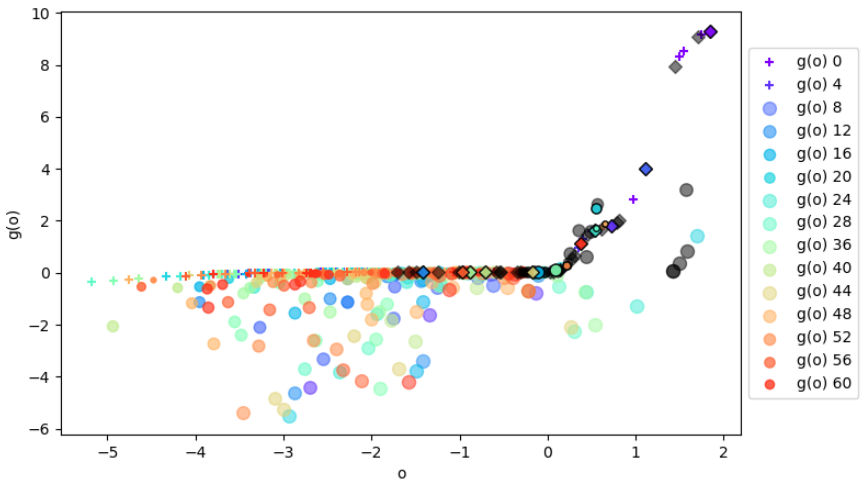

FIGURE 11 – Sorties $g_\theta(o)_t$ en fonction des vecteurs d'observations $o_{.,t}$ sur une trajectoire de test pour la méthode *NNTabu* NR avec les $g_\theta(o_a)_t$ contourées correspondant aux actions a_t sélectionnées pour $N = 32$ et $K = 8$

Comme la stratégie apprise par la méthode *NNTabu* permet bien de pallier le problème où la recherche se retrouve coincée en effectuant les mêmes actions en boucle en pénalisant ces dernières, alors celle-ci similaire la méthode de recherche locale *Tabu*. Nous pouvons d'ailleurs le vérifier avec la Table 2 où l'on trouve ses performances qui sont bien comparables à la méthode *Tabu*.

Méthodes	$N = 32$	$N = 64$
Tabu	0.755	0.759
NNTabu R	0.756	0.761
NNTabu NR	0.758	0.761

TABLE 2 – Résultats des tests pour la méthode *NNTabu* avec un réseau de neurones R ou NR et la méthode *Tabu* pour le problème NK-Landscape où $K = 8$

6.4 D'un choix déterministe à un choix stochastique

Jusque-là, les deux méthodes neuro-évolutionnaires que nous avons étudiées font un choix déterministe sur l'action à effectuer en fonction de la sortie $g_\theta(o)$ à laquelle on sélectionne toujours la plus grande valeur. Nous allons donc voir si en appliquant à la place la fonction $softmax$, nous pouvons mieux performer. Nous avons alors les méthodes *NNSoftmax(Tabu)* que nous allons comparer à leurs homologues déterministes.

En analysant la stratégie apprise par la méthode *NNSoftmax* dans la Figure 12, nous pouvons encore remarquer la présence de ce pic autour de 0.5 qui ici concerne les o pour un $\rho(o) < -1$ pouvant donner une probabilité choisie très proche de 1

(ce comportement est aussi observable en NR cf. Figure 13). C'est sans surprise dans cette zone légèrement améliorante que nous avons le plus d'actions qui sont choisies. À l'instar de la méthode *NN*, les actions légèrement améliorantes sont les plus appréciées. Les actions moyennement à très améliorantes sont celles qui sont les secondes plus appréciées avec une légère décroissance des $g\theta(o)$ après le pique autour de 0.5 suivie d'une monotonie mettant ces premières au même niveau. Quand $\rho(o) > -1$, se sont les meilleures actions possibles qui sont très valorisées afin d'accélérer la recherche en début de trajectoire. Les actions dégradantes sont, elles, très défavorisées dans tous les cas. Cependant, celles-ci peuvent tout de même être sélectionnées, comme un $softmax$ est appliqué sur g_θ(o), surtout quand $\rho(o) << 0$. Ainsi, même si en ne regardant que les $g_\theta(o) < 0$, nous avons l'impression d'avoir perdu la stratégie de descendre dans le score de *fitness*, la politique π_θ s'est adaptée pour ne pas trop la mettre en avant.

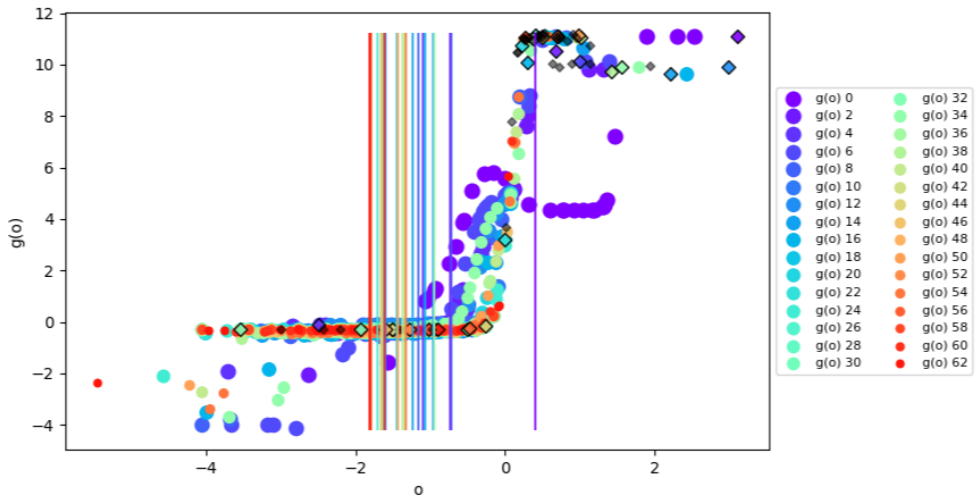

FIGURE 12 – Sorties $g_\theta(o)_t$ en fonction des vecteurs d'observations $o_{\cdot,t}$ sur une trajectoire de test pour la méthode *NNSoftmax* R avec les $g_\theta(o_a)_t$ contourées correspondant aux actions a_t sélectionnées et les barres verticales aux moyennes $\rho(o_{\cdot,t})$ pour $N = 32$ et $K = 8$

Quand cette fois-ci le réseau de neurones est NR, nous pouvons voir dans la Figure 13 que nous nous retrouvons qu'avec une seule courbe très claire qui correspond à celle pour des $\rho(o) < -1$ en R. De ce fait, le *NNSoftmax* en NR est légèrement plus lent en début de recherche qu'en R, car le second va préférer faire des sauts dans le score de *fitness* quand il en a la possibilité, quand le premier va toujours préférer l'améliorer doucement. Mis à part cette légère différence, les stratégies apprises en R et NR sont les mêmes pour la méthode *NNSoftmax*.

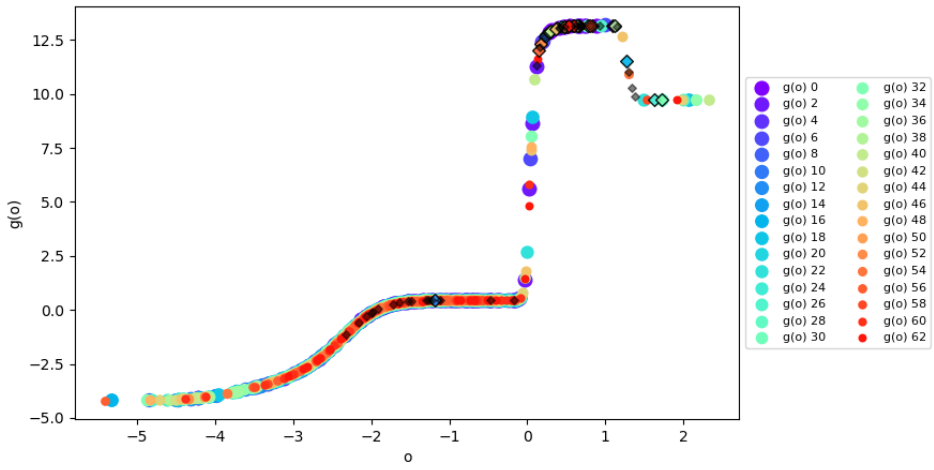

FIGURE 13 – Sorties $g_\theta(o)_t$ en fonction des vecteurs d'observations $o_{.,t}$ sur une trajectoire de test pour la méthode *NNSoftmax* NR avec les $g_\theta(o_a)_t$ contourées correspondant aux actions a_t sélectionnées et les barres verticales aux moyennes $\rho(o_{.,t})$ pour $N = 32$ et $K = 8$

Nous pourrions avoir l'impression, alors, que les méthodes *NN* et *NNSoftmax* sont équivalentes. Cependant, si l'on regarde la Table 3, le changement de fonction appliquée sur les sorties a permis d'améliorer le score de *fitness* moyen, et ce, en rendant la stratégie apprise plus flexible. Comme nous avions pu le voir précédemment, la méthode *NN* pouvait se retrouver coincée dans sa recherche. Comme celle-ci était radicale dans son choix en appliquant forcément l'action a correspondant au plus au grand $g\theta(o)$, alors la recherche se mettait à faire des actions en boucle pour essayer de sortir de ce maximum local, en vain. Mais en appliquant à présent un $softmax$, la probabilité de boucler sur les mêmes actions est pratiquement nulle. C'était d'ailleurs pour ce même problème que nous avions introduit la méthode *NNTabu*. Cependant, quand celle-ci est aussi rendue stochastique, nous pouvons voir qu'il n'y a pas d'amélioration. Effectivement, en faisant ce changement, nous essayons de pallier un problème déjà résolu en défavorisant les actions taboues.

Méthodes	$N = 32$	$N = 64$
$NN\ R$	0.733	0.737
$NN\ NR$	0.731	0.736
$NNSoftmax\ R$	0.741	0.748
$NNSoftmax\ NR$	0.741	0.747
$NNTabu\ R$	0.756	0.761
$NNTabu\ NR$	0.758	0.761
$NNTabuSoftmax\ R$	0.756	0.760
$NNTabuSoftmax\ NR$	0.755	0.760

TABLE 3 – Résultats des tests pour les méthodes *NN(Tabu)* et *NNSoftmax(Tabu)* avec un réseau de neurones R ou NR pour le problème NK-Landscape où $K = 8$

6.5 Passage à l'apprentissage par renforcemment

À l'aide de l'algorithme Neuro-LS modifié, nous avons pu obtenir des résultats très satisfaisants. Nous voulons maintenant vérifier s'il est possible de faire émerger de nouvelles stratégies et d'avoir au moins des résultats équivalents avec d'autres algorithmes que le CMA-ES. Nous allons donc analyser le comportement de l'algorithme Neuro-LS en remplaçant l'algorithme évolutionnaire CMA-ES par un algorithme d'apprentissage par renforcement. Comme les algorithmes que nous allons étudier vont devoir appliquer la fonction $softmax$ sur la sortie du réseau de neurones, nous ne considérerons que les méthodes $NN(Tabu)Softmax$ comme comparaison.

6.5.1 REINFORCE

Nous allons commencer avec l'algorithme REINFORCE et sa méthode *NNReinforce*. Cette dernière est équivalente à la méthode *NNSoftmax* dans le sens où seul le vecteur des observations o est donné en entrée du réseau de neurones et que sur sa sortie $g\theta(o)$ est appliquée la fonction $softmax$. La stratégie par celle-ci est plutôt simple, comme on peut le voir dans la Figure 14. Nous avons une courbe monotone en $]-\infty, -\varepsilon[$, fortement croissante en $[-\varepsilon, \varepsilon]$, puis monotone en $]\varepsilon, +\infty[$. ε un réel proche de 0. De ce fait, les actions améliorantes, peu importe l'amplitude d'amélioration, sont toujours les plus favorisées et inversement pour les actions dégradantes. Ainsi, plus il y a de valeurs positives dans o, plus les probabilités qu'une action dégradante soit tirée sont faibles. Ici, la stratégie ne prend pratiquement plus en compte $\rho(o)$ mais donne plus d'importance au ratio $o_- : o_+,\ \forall o_-, o_+ \in o$.

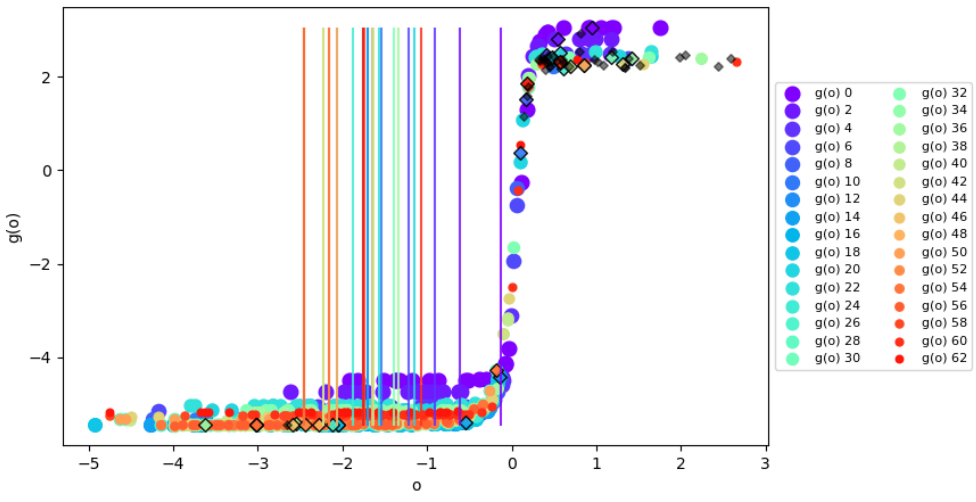

FIGURE 14 – Sorties $g_\theta(o)_t$ en fonction des vecteurs d'observations $o_{.,t}$ sur une trajectoire de test pour la méthode *NNReinforce* R avec les $g_\theta(o_a)_t$ contourées correspondant aux actions a_t sélectionnées et les barres verticales aux moyennes $\rho(o_{.,t})$ pour $N = 32$ et $K = 8$

Cela se vérifie en comparant *NNReinforce* en R et NR. Dans la Figure 15, nous avons bien une courbe avec le même aspect nous permettant d'affirmer que l'information $\rho(o)$ n'est plus importante.

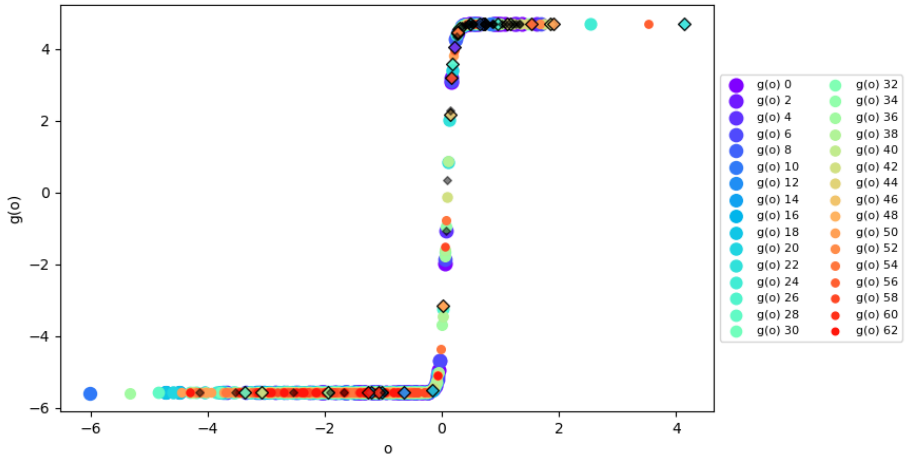

FIGURE 15 – Sorties $g_\theta(o)_t$ en fonction des vecteurs d'observations $o_{.,t}$ sur une trajectoire de test pour la méthode *NNReinforce* NR avec les $g_\theta(o_a)_t$ contourées correspondant aux actions a_t sélectionnées pour $N = 32$ et $K = 8$

Si nous observons la Figure 16, nous pouvons aussi ici remarquer que pour la méthode *NNReinforceTabu* en R, nous avons aussi une stratégie plus simple que *NNTabu*. À la différence de la courbe de *NNReinforce*, ici la courbe des actions non taboues et celle des actions taboues n'arrête pas leur croissance, donnant ainsi un avantage aux actions les plus améliorantes.

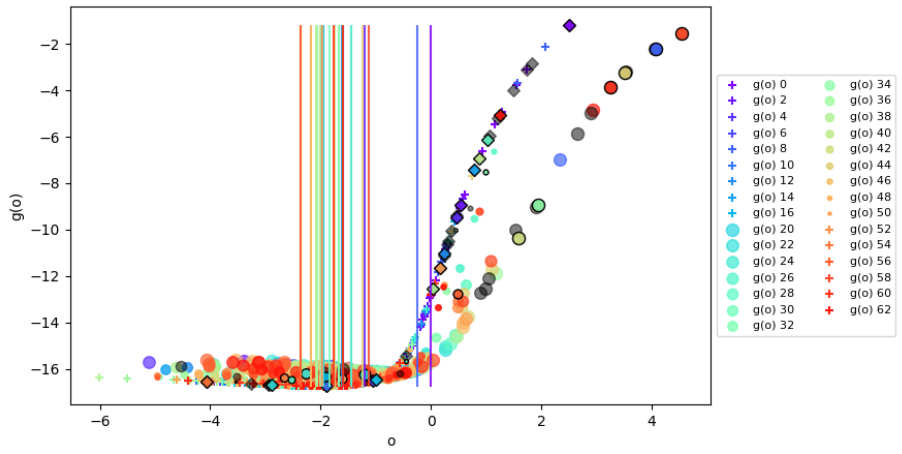

FIGURE 16 – Sorties $g_\theta(o)_t$ en fonction des vecteurs d'observations $o_{.,t}$ sur une trajectoire de test pour la méthode *NNReinforceTabu* R avec les $g_\theta(o_a)_t$ contourées correspondant aux actions a_t sélectionnées et les barres verticales aux moyennes $\rho(o_{.,t})$ pour $N = 32$ et $K = 8$

De plus, la première courbe croît plus vite que la seconde afin de donner plus d'importance aux actions non taboues à observations équivalentes des taboues.

Le comportement est un peu différent avec la méthode *NNReinforce* en NR. Dans la Figure 17, nous avons deux courbes logistiques avec celle des actions non taboues centrée à -0.75 et celle des actions taboues à -0.25 laissant cette fois-ci un avantage aux actions légèrement à moyennement dégradantes. La stratégie ne va donc préférer dégrader que légèrement le score de *fitness* quand la recherche se retrouve coincée dans un maximum local.

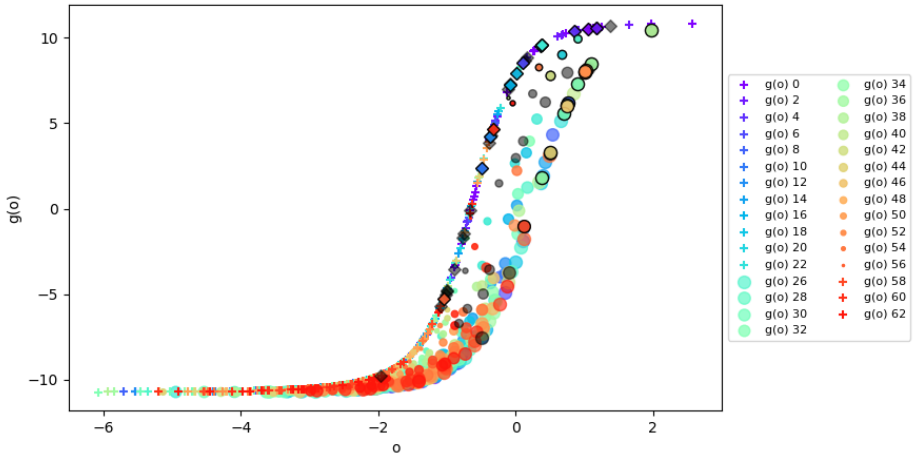

FIGURE 17 – Sorties $g_\theta(o)_t$ en fonction des vecteurs d'observations $o_{\cdot,t}$ sur une trajectoire de test pour la méthode *NNReinforceTabu* NR avec les $g_\theta(o_a)_t$ contourées correspondant aux actions a_t sélectionnées pour $N = 32$ et $K = 8$

Si nous comparons ces nouvelles stratégies avec celles du CMA-ES, nous remarquerons dans la Table 4 que le REINFORCE est légèrement moins performant que le CMA-ES. Cela peut s'expliquer par le fait que les stratégies apprises par les méthodes *NNReinforce(Tabu)* perdent en subtilité sur leur manière de valoriser les actions.

Méthodes	$N = 32$	$N = 64$
$NNSoftmax\ R$	0.741	0.748
$NNSoftmax\ NR$	0.741	0.747
$NNReinforce\ R$	0.740	0.745
$NNReinforce\ NR$	0.740	0.744
$NNTabuSoftmax\ R$	0.756	0.760
$NNTabuSoftmax\ NR$	0.755	0.760
$NNReinforceTabu\ R$	0.753	0.755
$NNReinforceTabu\ NR$	0.752	0.755

TABLE 4 – Résultats des tests pour les méthodes *NNSoftmax(Tabu)* et *NNReinforce(Tabu)* avec un réseau de neurones R ou NR pour le problème NK-Landscape où $K = 8$

6.5.2 PPO-KL

Après avoir étudié REINFORCE, nous allons maintenant passer au PPO-KL afin d'analyser, si, limiter la différence entre deux politiques π_θ et π'_θ peut nous permettre de découvrir de nouvelles stratégies. Pour la méthode *NNPPOKL* en R, nous pouvons observer dans la Figure 18 que nous avons des courbes du même aspect que celle de la méthode *NNReinforce*. Cependant, ici, l'information sur la moyenne $\rho(o)$ est bien prise en compte, comme nous pouvons le voir par le nombre de courbes et de la variation de croissance de ces dernières. Plus $\rho(o)$ est faible, plus la croissance des $g\theta(o)$ est forte ($g_\theta(o)_{.,0}$ à une croissance de ≈ 6 contre $g_\theta(o)_{.,56}$ à ≈ 10).

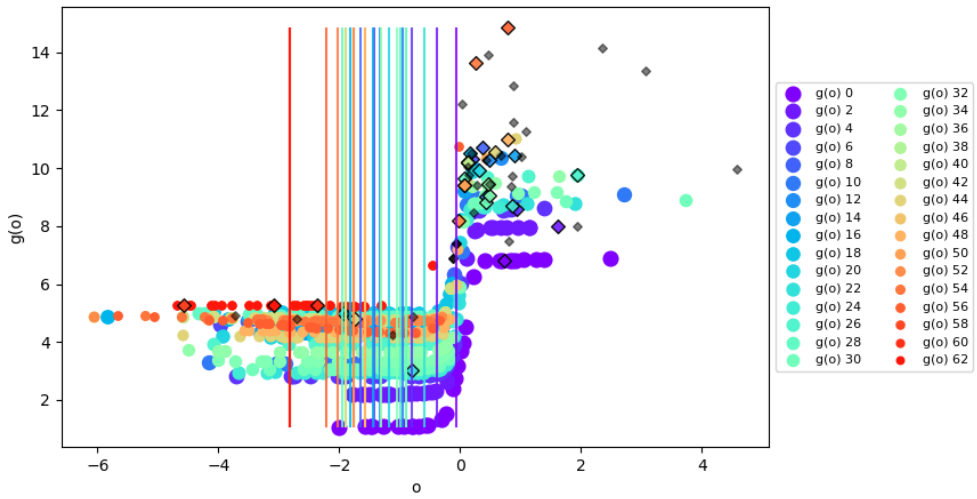

FIGURE 18 – Sorties $g_\theta(o)_t$ en fonction des vecteurs d'observations $o_{.,t}$ sur une trajectoire de test pour la méthode *NNPPOKL* R avec les $g_\theta(o_a)_t$ contourées correspondant aux actions a_t sélectionnées et les barres verticales aux moyennes $\rho(o_{.,t})$ pour $N = 32$ et $K = 8$

Dans cette stratégie, à la différence de *NNReinforce*, les actions dégradantes ont moins d'opportunités d'être tirées à cause de l'écart de valeur grandissant.

Pour la méthode *NNPPOKL* en NR, nous avons à présent dans la Figure 19 une courbe unique ressemblant à celle apprise par *NNReinforce*, cependant, celle-ci a sa croissance centrée sur -1 et a une seconde croissance, légère, entre 0.5 et 1.75 menant à une monotonie. Il était donc appris de fortement favoriser les actions moyennement à fortement améliorantes, suivies de près par les actions légèrement améliorantes et les actions légèrement dégradantes misent au même niveau. Les actions a correspondant à des $o_a < -1$ ne sont même plus considérées, car rares sont les cas où il n'y pas, au moins, une action légèrement dégradante. La faiblesse que peut avoir cette stratégie est qu'elle risque de coincer la recherche dans un maximum local et de ne pas lui donner le moyen d'en sortir en faisant un pas assez grand en arrière.

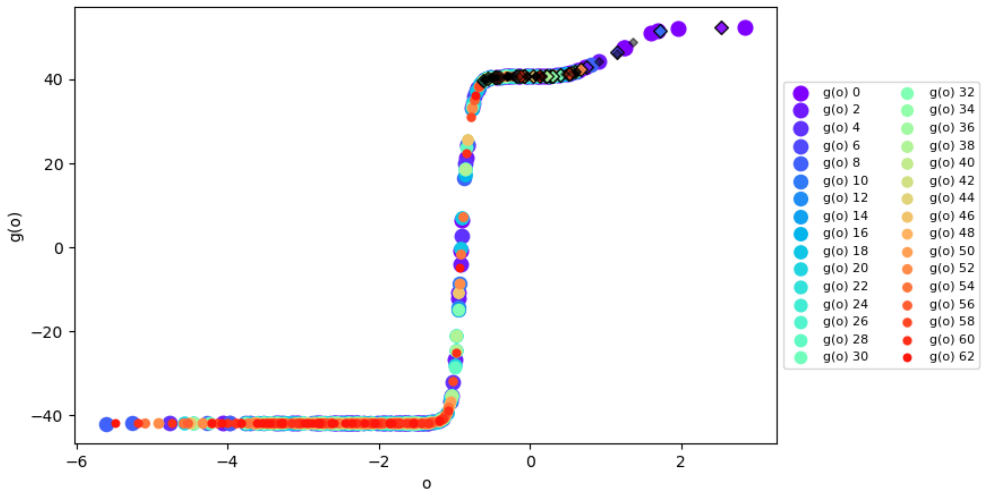

FIGURE 19 – Sorties $g_\theta(o)_t$ en fonction des vecteurs d'observations $o_{\cdot,t}$ sur une trajectoire de test pour la méthode *NNPPOKL* NR avec les $g_\theta(o_a)_t$ contourées correspondant aux actions a_t sélectionnées pour $N = 32$ et $K = 8$

Pour la méthode *NNPPOKLTabu* en R, nous pouvons remarquer dans la Figure 20 quelque chose d'assez surprenant : La forme des deux courbes sont très similaires à celle de la méthode *NN*. Une forme qui n'a d'ailleurs pas été apprise par *NNTabu*. Cependant, la différence est que c'est une fonction $softmax$ qui est appliquée ici sur $g_\theta(o)$ ce qui donne un peu plus de flexibilité dans le choix des actions. Il faut aussi remarquer que les deux courbes commencent à beaucoup se rapprocher, augmentant la probabilité qu'une action taboue soit sélectionnée.

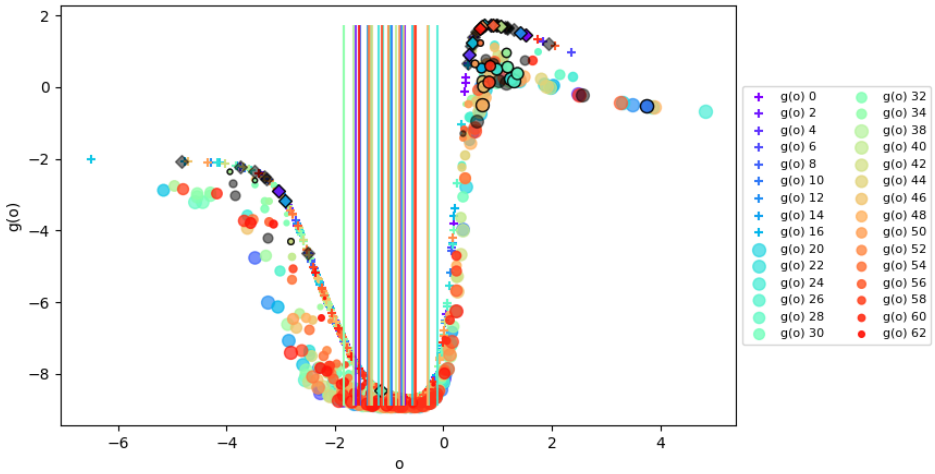

FIGURE 20 – Sorties $g_\theta(o)_t$ en fonction des vecteurs d'observations $o_{.,t}$ sur une trajectoire de test pour la méthode *NNPPOKLTabu* R avec les $g_\theta(o_a)_t$ contourées correspondant aux actions a_t sélectionnées et les barres verticales aux moyennes $\rho(o_{.,t})$ pour $N = 32$ et $K = 8$

D'autre part, le rapprochement des deux courbes pour la méthode *NNPPOKLTabu* en NR est tellement fort que la différence entre actions taboues et non taboues est pratiquement effacée. Cela nous laisse avec une stratégie quasi identique à celle de *NNPPOKL* où seulement la monotonie en $[-1-\varepsilon, 1+\varepsilon]$ est remplacée par une très légère croissance. Ainsi, les actions légèrement améliorantes et les actions légèrement dégradantes ne sont plus au même niveau avec un petit avantage en plus pour les premières.

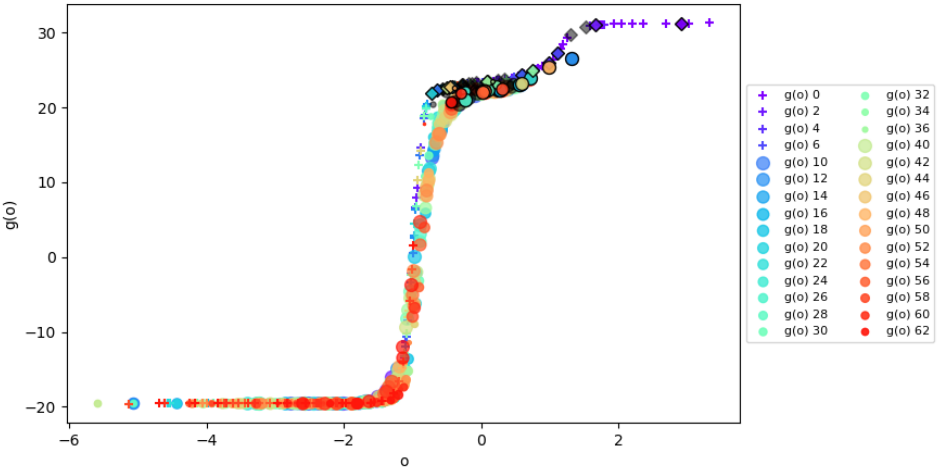

FIGURE 21 – Sorties $g_\theta(o)_t$ en fonction des vecteurs d'observations $o_{.,t}$ sur une trajectoire de test pour la méthode *NNPPOKLTabu* NR avec les $g_\theta(o_a)_t$ contourées correspondant aux actions a_t sélectionnées pour $N = 32$ et $K = 8$

Les faiblesses discutées sur les stratégies apprises par les méthodes *NNPPOKL(Tabu)* sont effectivement observables dans la Table 5. Pour la méthode *NNPPOKL* en R et $N = 64$, nous observons une baisse de 0.006 de score de *fitness* par rapport à la méthode *NNSoftmax* en R. Ceci est dû, comme nous pouvons le voir dans la Figure 22, à l'instabilité d'apprentissage des modèles $NNPPOKL$ en R et NR donnant des scores de *fitness* en apprentissage très variants. L'instabilité est davantage présente en moyenne en R. En NR, contrairement à en R, nous remarquerons tout de même que le meilleur modèle entrainé arrive à être aussi stable et est comparable en termes de score de *fitness* avec les modèles *NNSoftmax* qui ont, eux, un très faible écart-type.

Méthodes	$N = 32$	$N = 64$
$NNSoftmax\ R$	0.741	0.748
$NNSoftmax\ NR$	0.741	0.747
$NNPPOKL\ R$	0.740	0.742
$NNPPOKL\ NR$	0.740	0.746
$NNTabuSoftmax\ R$	0.756	0.760
$NNTabuSoftmax\ NR$	0.755	0.760
$NNPPOKLTabu\ R$	0.742	0.743
$NNPPOKLTabu\ NR$	0.747	0.754

TABLE 5 – Résultats des tests pour les méthodes *NNSoftmax(Tabu)* et *NNPPOKL(Tabu)* avec un réseau de neurones R ou NR pour le problème NK-Landscape où $K = 8$

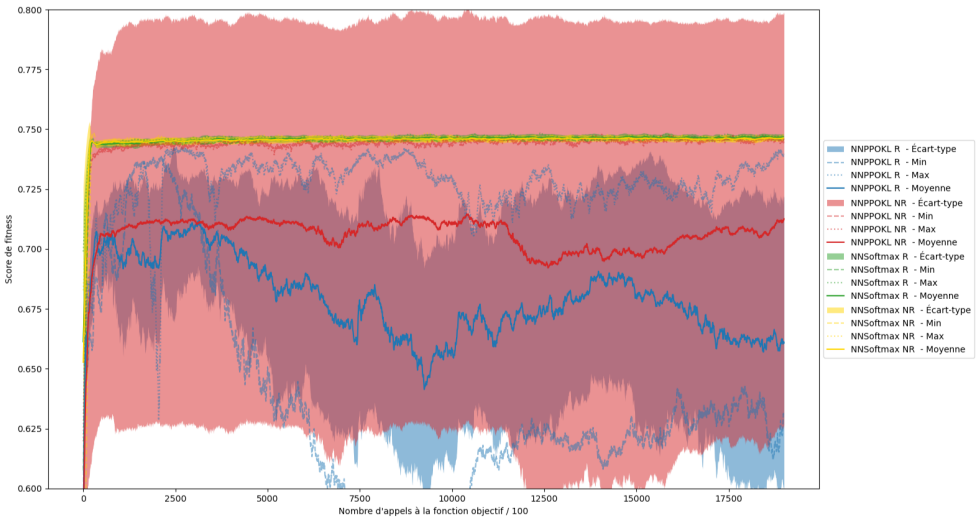

FIGURE 22 – Évolution du score de *fitness* en validation sur 10 *runs* modèles indépendants des méthodes *NNPPOKL* et *NNSoftmax* en N et NR en fonction du nombre d'appels à la fonction objectif / 100 pour N = 64

Si nous revenons à la Table 5 nous observerons que là, nous avons une encore plus forte perte en performance pour la méthode *NNPPOKLTabu* comparer à la méthode *NNSoftmaxTabu*, et ce, d'autant plus en R. Ceci est observable encore une fois dans la Figure 23 avec l'instabilité moyenne des modèles *NNPPOKLTabu* en R toujours présente. Les modèles en NR sont plutôt stables jusqu'autours du 1300000^{e} appel à la fonction objectif où en moyenne le score de *fitness* se détériore fortement. Ce qui est intéressant ici est que ces modèles ont pu être stables sans pour autant atteindre les performances en validation. La distinction faible entre actions taboues et actions non taboues pourrait s'expliquer par le fait que le réseau de neurones n'arrive pas ici à bien distinguer leurs impacts différents sur la recherche.

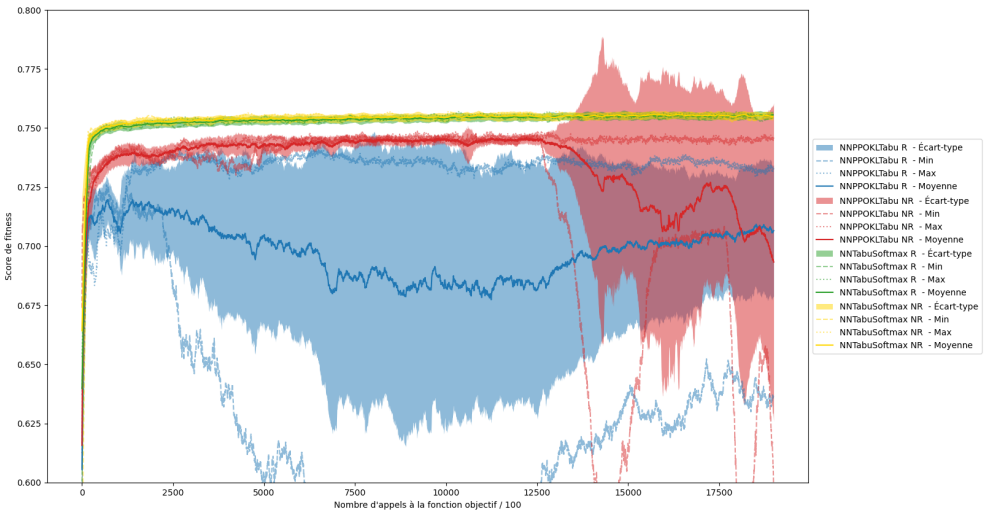

FIGURE 23 – Évolution du score de *fitness* en validation sur 10 *runs* modèles indépendants des méthodes *NNPPOKL* et *NNSoftmax* en N et NR en fonction du nombre d'appels à la fonction objectif / 100 pour N = 32

Nous pouvons en conclure que l'algorithme PPO-KL n'a pas réussi à donner de performances satisfaisantes comparer à l'algorithme CMA-ES. En utilisant le PPO-KL, soit la recherche est instable, soit les informations données au réseau de neurones ne sont pas correctement interprétées. J'ai donc essayé de faire plusieurs modifications comme modifier la taille du réseau de neurones, le nombre d'*epoch*, la fonction d'activation (cf. Table 7) et de ne considérer que les couples (s_t, a_t) pour un t inférieur ou égal au tour t_{max} où l'on trouve le score de *fitness* maximal r_{max} de la trajectoire (cf. Table 8). Cependant, tous ces changements n'ont pas permis d'être au moins aussi performant qu'avec l'algorithme CMA-ES.

6.6 Rangs et normalisation

Dans cette dernière expérience, nous allons étudier la méthode *NNRanked* afin de déterminer si quand les observations sont normalisées, nous arrivons à avoir une stratégie équivalente à celle de la méthode *NN*. Sur la Figure 24 pour la méthode *NNRanked* en R, nous avons pour des $\rho(o) < -0.5$, une décroissance en $[-1, \varepsilon]$, et pour des $\rho(o) > -0.5$, nous avons d'abord une très légère croissance en $[-1, -0.6+\varepsilon]$ qui est suivi d'une décroissance en $[-0.6+\varepsilon, -\varepsilon]$. Enfin, $\forall\rho(o)$ nous avons une forte croissance commençant à $\pm\varepsilon$ pour qu'en suite les courbes fassent plateau jusqu'à 1. Comme la méthode *NNRanked* est la version normalisée de la méthode *NN*, c'est la fonction $argmax$ qui appliquée sur les sorties $g_\theta(o)$. Ainsi, ce sont les actions a ayant une observation o_a supérieure ou égale à la médiane qui sont sélectionnées. De plus, le fait de mettre sous forme de rang et de normaliser les o fait en sorte que même s'il n'y a qu'une seule action améliorante, celle-ci sera forcément sélectionnée, peu importe son niveau d'amélioration à la différence de la méthode *NN*. Dans le cas où il n'y aurait aucune action améliorante, alors ce sera l'action la plus dégradante qui sera effectuée.

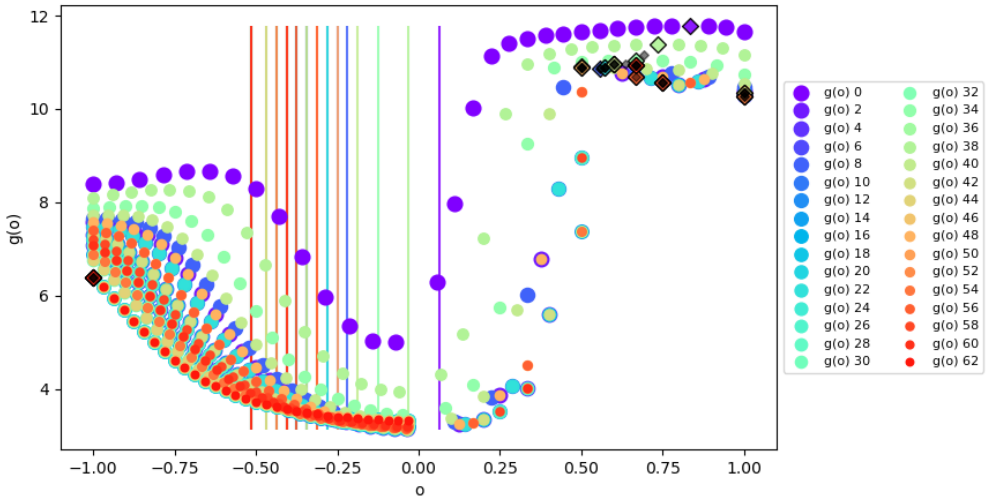

FIGURE 24 – Sorties $g_\theta(o)_t$ en fonction des vecteurs d'observations $o_{\cdot,t}$ sur une trajectoire de test pour la méthode *NNRanked* R avec les $g_\theta(o_a)_t$ contourées correspondant aux actions a_t sélectionnées et les barres verticales aux moyennes $\rho(o_{\cdot,t})$ pour $N = 32$ et $K = 8$

Si nous regardons la stratégie apprise par la méthode *NNRanked* en NR dans la Figure 25, nous retrouvons une courbe généralisant l'aspect de celles en R. Il a donc émergé une stratégie très similaire où ce sont les actions les plus améliorantes qui sont sélectionnées suivi par l'action la plus dégradante.

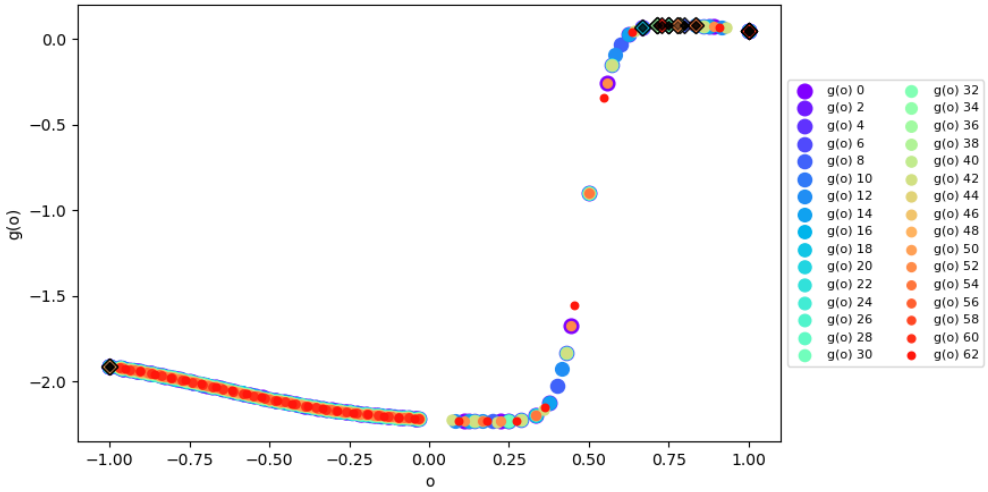

FIGURE 25 – Sorties $g_\theta(o)_t$ en fonction des vecteurs d'observations $o_{.,t}$ sur une trajectoire de test pour la méthode *NNRanked* NR avec les $g_\theta(o_a)_t$ contourées correspondant aux actions a_t sélectionnées pour $N = 32$ et $K = 8$

La méthode *NNRanked* a pu apprendre une stratégie très proche de celle apprise par la méthode *NN*, et ceux, même en n'ayant pas l'information sur l'amplitude réelle d'amélioration ou des dégradations des actions. Nous pouvons aussi retrouver cette similarité dans le score de performance comme nous pouvons le voir dans la Table 6 où la méthode *NNRanked* en R et en NR n'atteint pas un score de *fitness* moyen en test inférieurs à ceux de la *NN*.

Méthodes	$N = 32$	$N = 64$
$NN\ R$	0.733	0.737
$NN\ NR$	0.731	0.736
$NNRanked\ R$	0.734	0.738
$NNRanked\ NR$	0.731	0.738

TABLE 6 – Résultats des tests pour les méthodes *NN* et *NNRanked* avec un réseau de neurones R ou NR pour le problème NK-Landscape où $K = 8$

7 Autres perspectives

Plusieurs modifications peuvent encore être appliquées à l'algorithme Neuro-LS afin de faire émerger encore d'autres stratégies et possiblement améliorer ses performances. En voici quelques-unes :

— Donner au réseau de neurones plus de contexte sur l'état de la recherche comme le numéro de tour $t \in [0..T[$, le score de *fitness* au tour précédent ou le score de *fitness* maximal rencontré.

— Mettre et normaliser le vecteur d'observations o comme dans la méthode $NNRanked$ pour les méthodes des algorithmes d'apprentissage profond et surtout pour le PPO-KL pour le rendre plus stable dans sa recherche.

— Essayer d'obtenir la stabilité des méthodes utilisant PPO-KL en agrandissant plus l'architecture du réseau de neurones par son nombre de couches ou de neurones.

— L'algorithme PPO-KL peut aussi être modifié en lui-même en remplaçant la pénalité KL par le concept d'objectif clippé qui est un autre moyen d'éviter une forte divergence entre deux politiques π_θ et π'_θ [9].

— Utiliser un algorithme d'apprentissage par renforcement multi-agents en coopération afin que ces derniers puissent se communiquer leurs bons choix et leurs erreurs [12].

8 Conclusion

Nous avons étudié plusieurs méthodes différentes associées aux algorithmes CMA-ES, REINFORCE et PPO-KL sur deux réseaux de neurones différents où l'un prend en compte la moyenne des observations faites afin d'optimiser le problème des paysages NK. À travers ces méthodes, plusieurs stratégies plus ou moins subtiles dans la valorisation des actions ont pu émerger. Toutes ont réussi à être plus performantes que les méthodes de recherche locale classiques (hors $Tabu$). Nous avons d'abord les méthodes *NN* R/NR et *NNSoftmax* basées sur l'algorithme CMA-ES qui choisissent pratiquement toujours dans cet ordre, d'abord les actions légèrement améliorantes, puis moyennement à très améliorantes et enfin dégradantes (la variation peut être due à la valeur de $\rho(o)$ ou des probabilités en sortie de la fonction $softmax$). La méthode *NNSoftmax* en NR, elle, préfère laisser moins de chances d'être sélectionnées aux actions légèrement à moyennement dégradantes et pratiquement aucune chance pour les très dégradantes. De plus, en ajoutant plus de flexibilité dans le choix des actions à l'aide de fonction $softmax$ nous avons pu gagner légèrement en performance. Cependant, la méthode ayant pu faire émerger une stratégie la plus subtile et la plus performante est celle de *NNTabu* qui permet de ne pas laisser la recherche coincée dans un maximum local. L'application de la fonction $softmax$ pour cette méthode n'avait donc pas d'intérêt supplémentaire. Les méthodes basées sur les algorithmes REINFORCE et PPO-KL n'ont, quant à elles, pas pu aussi bien performer que celles basées sur l'algorithme CMA-ES. L'analyse des stratégies qui en n'ont émergées nous a montré que ces méthodes n'arrivaient pas correctement ou trop pauvrement à interpréter les informations données au réseau de neurones. Il faudrait donc trouver d'autres manières d'exprimer ces informations ou de revoir la structure du réseau de neurones en lui-même. L'incorporation de l'information de moyenne dans le réseau de neurones a pu pour certaines méthodes permettre d'apporter de la subtilité dans les stratégies apprises en faisant du cas par cas. Comme dit dans la Section 7, il reste encore plein de perspectives à explorer pour pouvoir découvrir de nouvelles stratégies émergeant de l'algorithme Neuro-LS modifié de différentes manières.

J'ai pu, au cours de mon stage, approfondir mes connaissances à travers la découverte des algorithmes CMA-ES, REINFORCE et PPO-KL, et de l'analyse des expériences que j'ai effectuées. Cela m'a aussi permis de mieux maîtriser la librairie PyTorch. J'espère donc pouvoir, à l'avenir, continuer de travailler sur des sujets de recherche en optimisation et d'en apprendre davantage.

9 Annexes

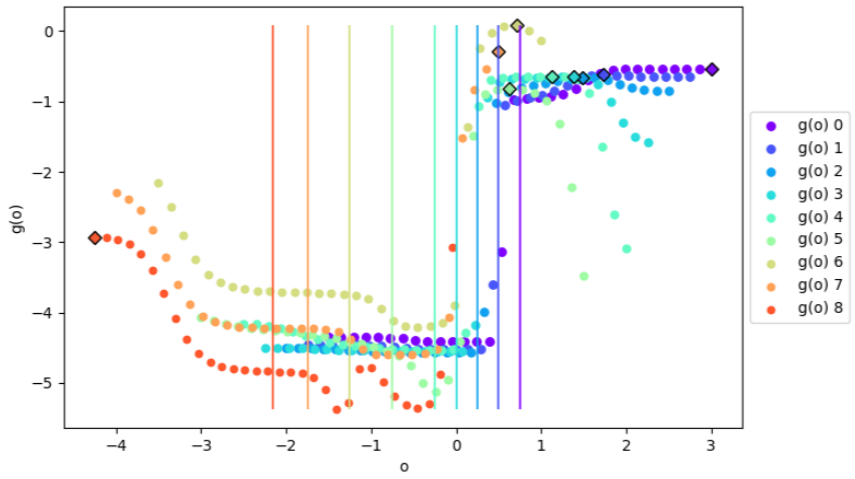

FIGURE 26 – Sorties $g_\theta(o)_t$ en fonction des vecteurs d'observations $o_{.,t}$ générés pour la méthode NN avec les $g_\theta(o_a)_t$ contourées correspondant aux actions a_t sélectionnées et les barres verticales aux moyennes $\rho(o_{.,t})$ pour $N = 32$ et $K = 8$

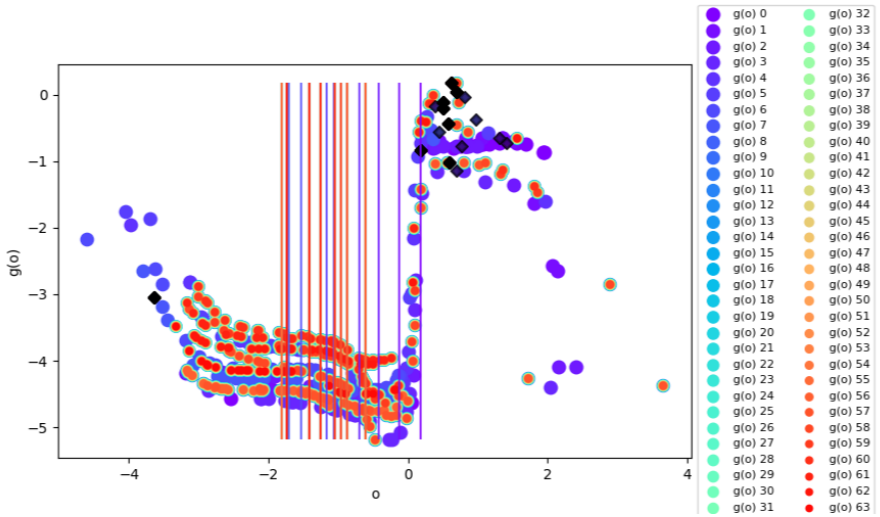

FIGURE 27 – Cas particulier des sorties $g_\theta(o)_t$ en fonction des vecteurs d'observations $o_{.,t}$ générés pour la méthode NN avec les $g_\theta(o_a)_t$ contourées correspondant aux actions a_t sélectionnées et les barres verticales aux moyennes $\rho(o_{.,t})$ pour $N = 32$ et $K = 8$

TABLE 7 – Résultats des tests pour les méthodes *NNPPOKL(Tabu)* avec un réseau de neurones R ou NR et en faisant la taille du réseau de neurones, le nombre d'*epoch* et la fonction d'activation pour le problème NK-Landscape où $N = 32$ et $K = 8$

Première partie de la Table 1/2					
Couches	Epoch	Méthode	Activation	Remix	Fitness
10,5	1	Softmax	Tanh	R	0.736
				NR	0.741
		Tabu	Tanh	R	0.748
				NR	0.749
	5	Softmax	Tanh	R	0.738
				NR	0.737
			Sigmoid	R	0.735
				NR	0.738
			ReLU	R	0.739
				NR	0.708
			LeakyReLU	R	0.738
				NR	0.740
		Tabu	Tanh	R	0.746
				NR	0.742
			Sigmoid	R	0.738
				NR	0.743
			ReLU	R	0.753
				NR	0.750
			LeakyReLU	R	0.742
				NR	0.749
	10	Softmax	Tanh	R	0.740
				NR	0.740
			Sigmoid	R	0.741
				NR	0.741
			ReLU	R	0.738
				NR	0.740
			LeakyReLU	R	0.738
				NR	0.740
		Tabu	Tanh	R	0.742
				NR	0.747
			Sigmoid	R	0.749
				NR	0.747
			ReLU	R	0.751
				NR	0.749
			LeakyReLU	R	0.751
				NR	0.751

Seconde partie de la Table 2/2					
Couches	Epoch	Méthode	Activation	Remix	Fitness
10,10	10	Softmax	Tanh	R NR	0.737 0.741
		Tabu	Tanh	R NR	0.739 0.741
20,10	10	Softmax	Tanh	R NR	0.740 0.741
		Tabu	Tanh	R NR	0.738 0.741

Couches	Epoch	Méthode	Activation	Remix	Fitness
10,5	10	Softmax	Tanh	R NR	0.737 0.739

TABLE 8 – Résultats des tests pour la méthode *NNPPOKL* avec un réseau de neurones R ou NR et en faisant la taille du réseau de neurones, le nombre d'*epoch* et la fonction d'activation pour le problème NK-Landscape où $N = 32$ et $K = 8$

Références

[1] Muhamet Kastrati, Marenglen Biba, International Journal of Electrical and Computer Engineering (IJECE)Vol.11, No.1, February2021, pp. 716 727, 2021. doi : 10.11591/ijece.v11i1.pp716-727

[2] Olivier Goudet, Mohamed Salim Amri Sakhri, Adrien Goëffon, Frédéric Saubion, *Emergence of new local search algorithms with neuro-evolution*, EvoCOP2024, 2024. HAL Id : hal-04457723

[3] Adrien Goëffon, *Modèles d'abstraction pour la résolution de problèmes combinatoires*, Thèse d'habilitation à diriger des recherches, Université d'Angers, 2014. Lien : https://leria-info.univ-angers.fr/ adrien.goeffon/Publi/Goeffon-HDR-court.pdf

[4] Csaszar, Felipe A., *A note on how NK landscapes work*, Conference : Evolutionary Computation, 1996., Proceedings of IEEE Journal of Organization Design, ISSN 2245-408X, Springer, Cham, Vol. 7, Iss. 15, pp. 1-6, 2018. doi : 10.1186/s41469-018-0039-0

[5] Nikolaus Hansen, Andreas Ostermeier, *Adapting arbitrary normal mutation distributions in evolution strategies : The covariance matrix adaptation*, Conference : Evolutionary Computation, 1996., Proceedings of IEEE International Conference on, 1996. doi : 10.1109/ICEC.1996.542381

[6] Masahiro Nomura, Masashi Shibata, *cmaes : A Simple yet Practical Python Library for CMA-ES*, 2024. doi : 10.48550/arXiv.2402.01373

[7] R.J. Williams, *Simple statistical gradient-following algorithms for connectionist reinforcement learning*, Mach Learn 8, 229–256, 1992. doi : 10.1007/BF00992696

[8] Ryan Peck, Louis Renaux, *A Review of REINFORCE Algorithms*, 2019. Lien : https://ryanpe05.github.io/EE227C_Final.pdf

[9] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov, *Proximal Policy Optimization Algorithms*, OpenAI, 2017. doi : 10.48550/arXiv.2402.01373

[10] Skiena S. Steven, *The Algorithm Design Manual (2nd ed.)*, Springer Science+Business Media, 2010. ISBN : 978-1-849-96720-4

[11] Sara Tari, Matthieu Basseur, Adrien Goëffon, *On the use of* $(1, \lambda)$*-evolution strategy as efficient local search mechanism for discrete optimization : a behavioral analysis*, Nat Comput 20, 345–361, 2001. doi : 10.1007/s11047-020-09822-2

[12] Craig Boutilier, *Planning, learning and coordination in multiagent decision processes*, TARK '96 : Proceedings of the 6th conference on Theoretical aspects of rationality and knowledge Pages 195–210, 1996. ISBN : 1558604179