Self Organizing Map

par Grégory Moutote et Nicolas Guruphat

3 Étude “théorique” de cas simples

3.1 Influence de η

Dans le cas où η = 0 quelle sera la prochaine valeur des poids du neurone gagnant ?

Dans ce cas là, le delta sera de 0 car le taux d'apprentissage est facteur de toute la formule. Le poids ne changera donc pas.

Même question dans le cas où η = 1

Si le taux d'apprentissage est égal à 1, $(x_i - w_{ij})$ tendra vers 0 (car nous travaillons sur le neuronne gagnant), donc le delta tendra également vers 0 au fur et à mesure que le neurone gagnant se rapprochera de l'entrée.

Dans le cas où $η ∈]0, 1[$ (paramétrisation “normale”) où se situera le nouveau poids par rapport à W∗ et X en fonction de η (formule mathématique simple ou explication géométrique) ?

$$\Delta w_{ij} = \eta e^{-\frac{||j-j^||^2_c}{2\sigma^2}} (x_i - w_{ij}) $$ En considérant que $j = j^$ devient alors :

$$ \Delta w_{ij} = \eta \times 1 \times (x_i - w_{ij}) \Leftrightarrow \Delta w_{ij} = \eta (x_i - w_{ij}) $$

Sachant que $\eta \in]0;1[$ et que $(x_i - w_{ij}) {\rightarrow} 0$, on peut dire que $\Delta w_{ij}\rightarrow 0$ avec une évolution semblable de e^{-x}. Donc, l'évolution du poids w sera logarithmique.

Que va-t-il se passer si η > 1 ?

Si $\eta > 1$, les poids risquent de permettre un déplacement du neurone qui dépassera l'entrée. On peut donc dire qu'il "sur-réagira" à l'entrée en lui donnant un effet sur son poids trop grand par rapport à ce qu'il devrait être.

3.2 Influence de σ

Si σ augmente, les neurones proches du neurone gagnant (dans la carte), vont-ils plus ou moins apprendre l'entrée courante ?

Équation	Variation
σ	$\nearrow$
$2\sigma^2$	$\nearrow$
$\frac{||j-j^*||_c^2}{2\sigma^2}$	$\searrow$

Équation	Variation
$e^{\frac{\|j-j^*\|_c^2}{2\sigma^2}}$	$\searrow$
$\frac{1}{e ^{\frac{\|j-j^\|_c^2}{2\sigma^2}}} \Leftrightarrow e ^{-\frac{\|j-j^\|_c^2}{2\sigma^2}}$	$\nearrow$
Δw	$\nearrow$

Comme le démontre ce tableau, les neurones proches du neurone gagnant seront plus modifiés si σ augmente.

Si σ est plus grand, à convergence, l'auto-organisation obtenue sera-t-elle donc plus "resserrée" (i.e. une distance plus faible entre les poids des neurones proches) ou plus "lâche" ?

Si σ est plus grand, on aura, à convergence, une auto-organisation plus resserrée. En effet, l'augementation de σ implique une augmentation des poids et donc une attraction plus forte du neurone gagnant, ce qui concentrera les neurones en un réseau plus serré.

Quelle mesure (formule mathématique qui sera à implémenter dans la section 4) pourrait quantifier ce phénomène et donc mesurer l'influence de σ sur le comportement de l'algorithme ?

La mesure nous permettant de quantifier l'impact de σ sur le comportement de l'algorithme est la dérivée de la fonction calculant Δw en fonction de σ. En effet, cette dérivée nous indiquera les variations de l'évolution.

$$ \Delta w(\sigma)= \eta e^{-\frac{\|j-j^\|^2_c}{2\sigma^2}} (x_i - w_{ij})\newline \Rightarrow \Delta w'(\sigma)=-\eta(x_i - w_{ij})\times\|j-j^\|^2_c\sigma^{-3}e^{\frac{-\|j-j^*\|^2_c}{2\sigma^2}} $$

Pour calculer le résultat de cette évolution, nous faisons la somme des différence de poid entre les neuronnes adjacents. Si on prend sy la taille de la carte sur l'axe y et sx la taille de la carte sur l'axe x,

$$ \sum_{n=0}^{sx-1} \sum_{m=0}^{sy} \|j_{(n;m)}-j_{(n+1;m)}\|+ \sum_{n=0}^{sx} \sum_{m=0}^{sy-1} \|j_{(n;m)}-j_{(n;m+1)}\| $$

3.3 Influence de la distribution d'entrée

Prenons le cas très simple d'une carte à 1 neurone qui reçoit deux entrées X1 et X2

Si X1 et X2 sont présentés autant de fois, vers quelle valeur convergera le vecteur du poids du neurone (en supposant un η faible - pour ne pas avoir à tenir compte de l'ordre de présentation des entrées - et suffisamment de présentations - pour négliger l'influence de l'initialisation des poids) ?

La valeur du neurone se trouvera entre celles des deux entrées (à équi-distance)

Même question si X1 est présenté n fois plus que X2.

```
x1|---------N----------------|x2
   <-------> <-------------->
    1/(n+1)      n/(n+1)
```

Avec N le neurone. La distance entre N et les entrées sera inversement proportionelle à la représentation de ces mêmes entrées. Plus leur représentation sera grande, plus le neurone en sera proche.

On voit sur ce schéma que la valeur du poid de N est la moyenne pondérée des poids de x1 et x2 (la pondération représentant le nombre d'occurrence)

Même question dans le cas d'entrées provenant d'une base de données quelconque.

Le poid du neurone sera le barycentre des entrées. En effet, le barycentre représente le résultat pondéré de la position de vecteurs.

$w=\frac{k_1\times x_1 + k_2 \times x_2 + k_3 \times x_3}{k_1+k_2+k_3}$

Avec :

- k_i le nombre d'occurence de l'entrée i
- x_i le poids de l'entrée i

Reconsidérons maintenant le cas d'une carte avec plusieurs neurones en se focalisant sur un neurone quelconque. Quels exemples apprend ce neurone et avec quelle "force" ?

Si ce neurone quelconque n'est pas le neurone gagnant, il apprendra ce vers quoi le neuronne gagnant tend, avec une force inversement proportionnelle à sa distance avec le neurone gagnant.

En déduire comment vont se répartir les neurones d'une carte à plusieurs neurones recevant des données d'une base d'apprentissage en fonction de la densité des données. Pour rappel, la mesure de quantification vectorielle permet de mesurer ce phénomène.

Si on prend un espace avec une proportion non uniforme d'entrée, il y aura plus de neurones gagnants dans la zone plus dense. Comme il y aura plus de neurones gagnants, plus de neuronnes seront attirés dans la zone plus dense. Toutefois, il restera un certain nombre de tirages dans la zone moins dense, mais qui sera moindre par rapport à la zone plus dense. Il y aura donc une répartion en proportion de la densité. Cela implique que l'espace entre les neurones sera optimal une fois que le réseau aura convergé.

Dans un espace avec un proportion uniforme d'entrée, la logique sera la même. En effet, la proportion de neurones gagnant par "zone" sera dorénavant uniforme et l'espace sera encore une fois réparti optimalement entre les neurones.

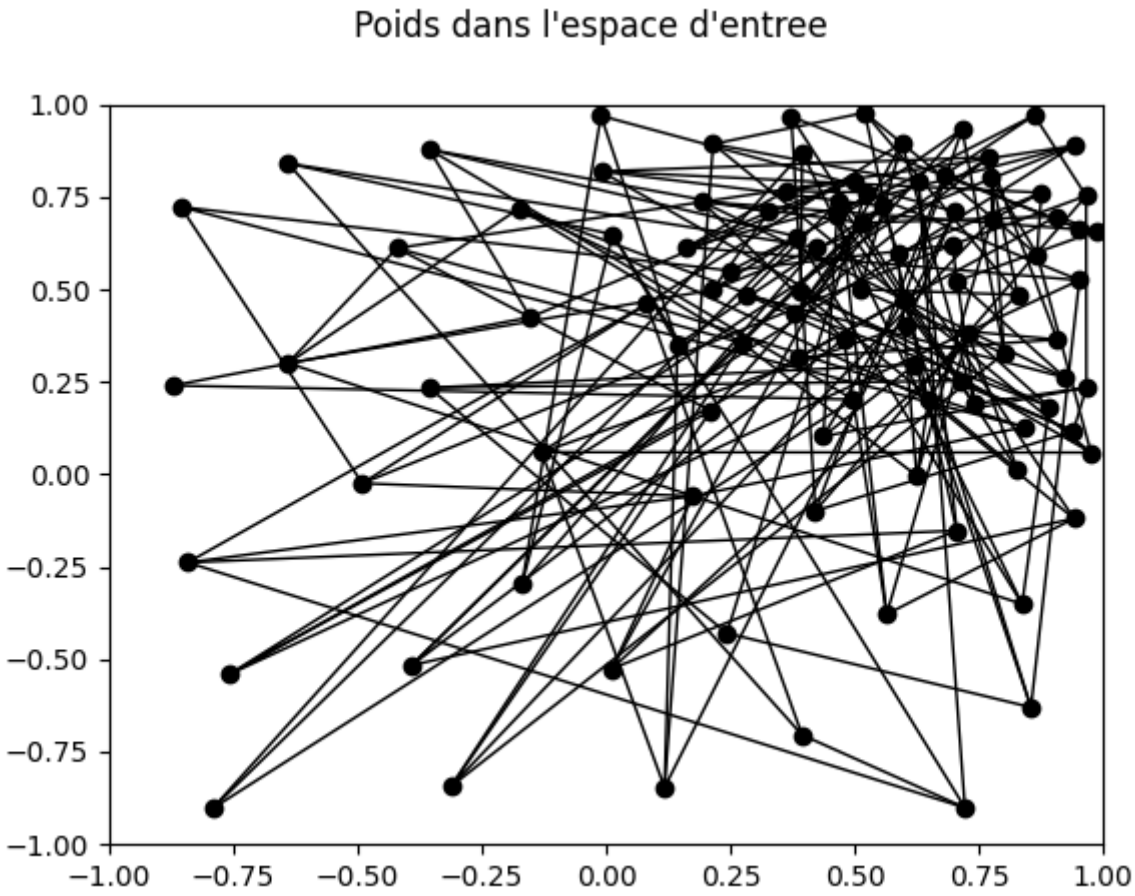
4

4.3 Analyse de l'algorithme

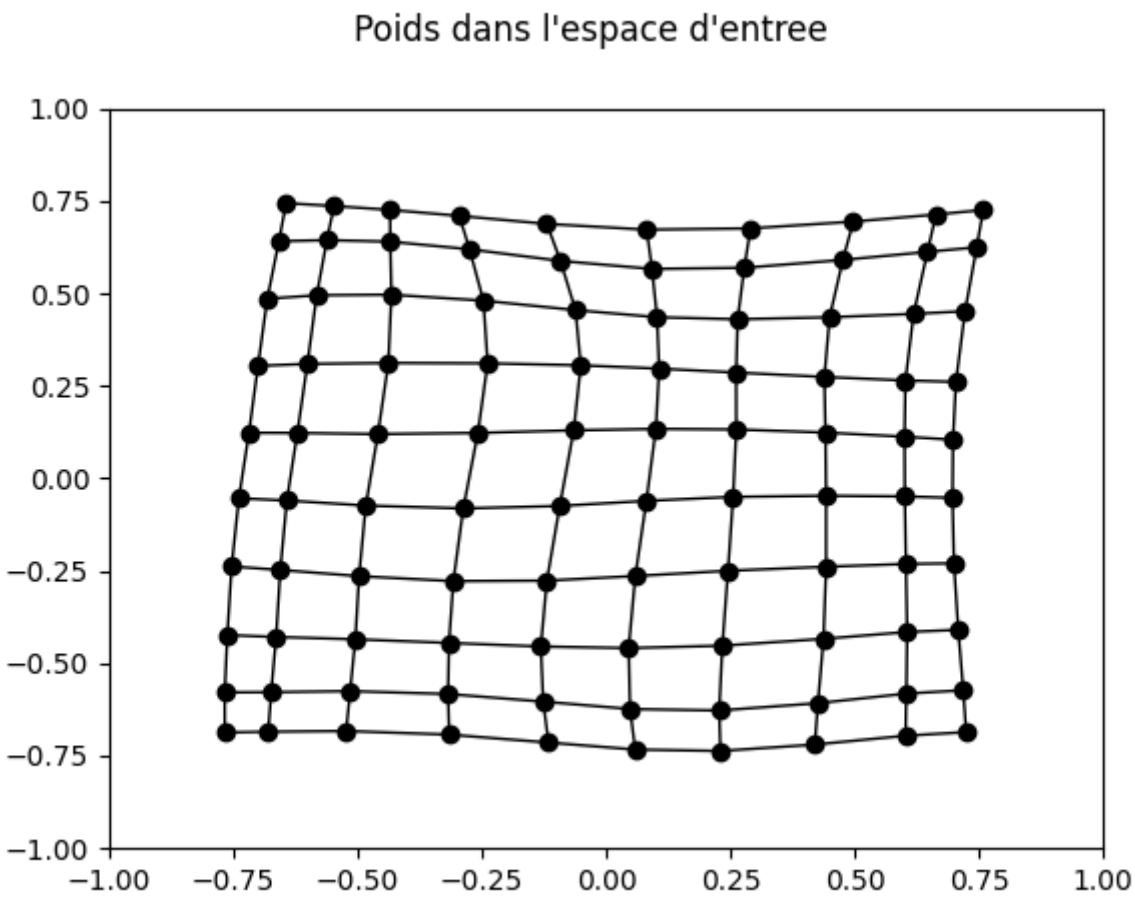
Pour cette partie, des cartes avec des η, σ et N ont été générés, chaque triplet a été exécuté 10 fois dont les moyennes de la MSE et de l'espacement neuronal ont été enregistrées ainsi que le maximum et le minimum pour ces dernières. L'ensemble des triplets correspond au produit cartésien de : $${0.0, 0.001, 0.05, 0.125, 0.25, 0.5, 0.75, 1.0, 2.0} \space pour \space \eta \newline {0.1, 0.2, 0.3, 0.4, 1.0, 1.4, 2.0, 3.0, 4.0} \space pour \space \sigma \newline {u_0...u_{48}} \space avec \space u_{n+1} = u_n + 1000 \space et \space u_0 = 1000 \space pour \space N$$

Variation de σ

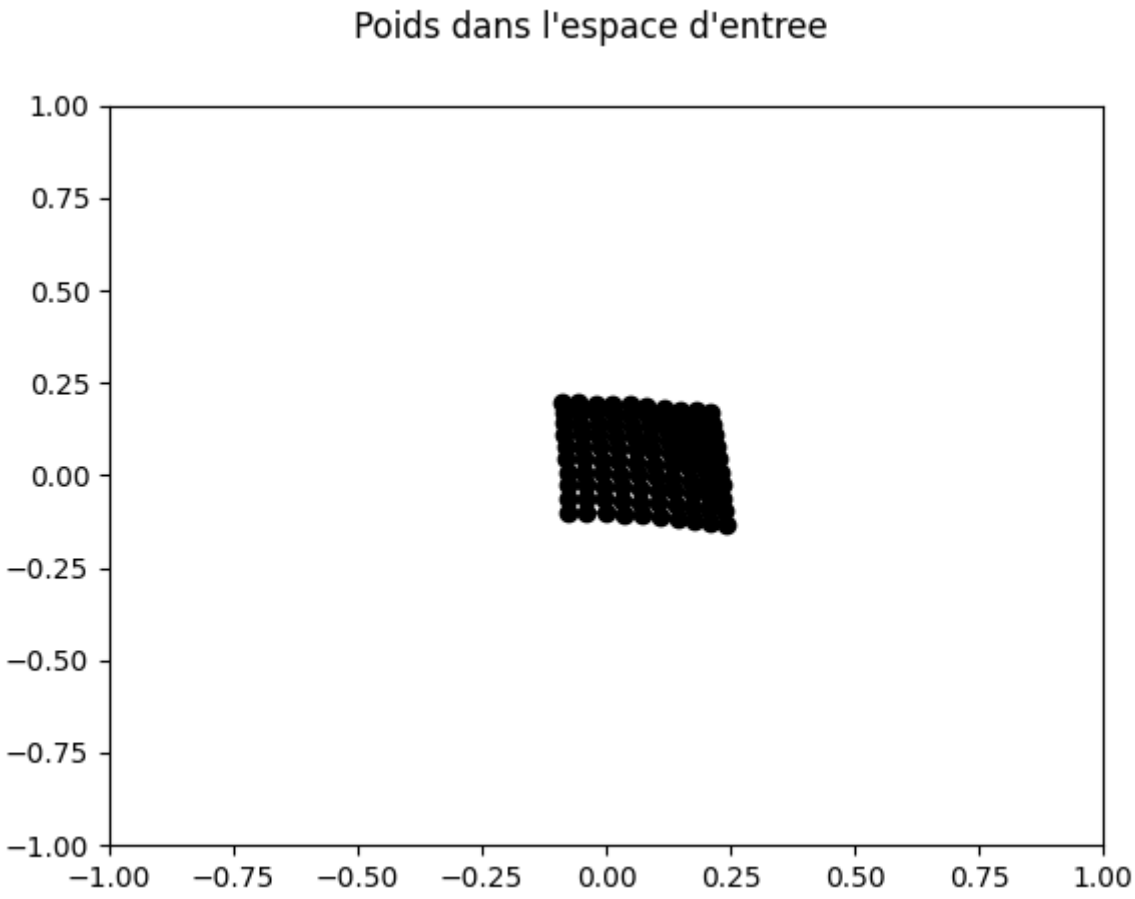
sigma = 0.1



sigma = 1.4

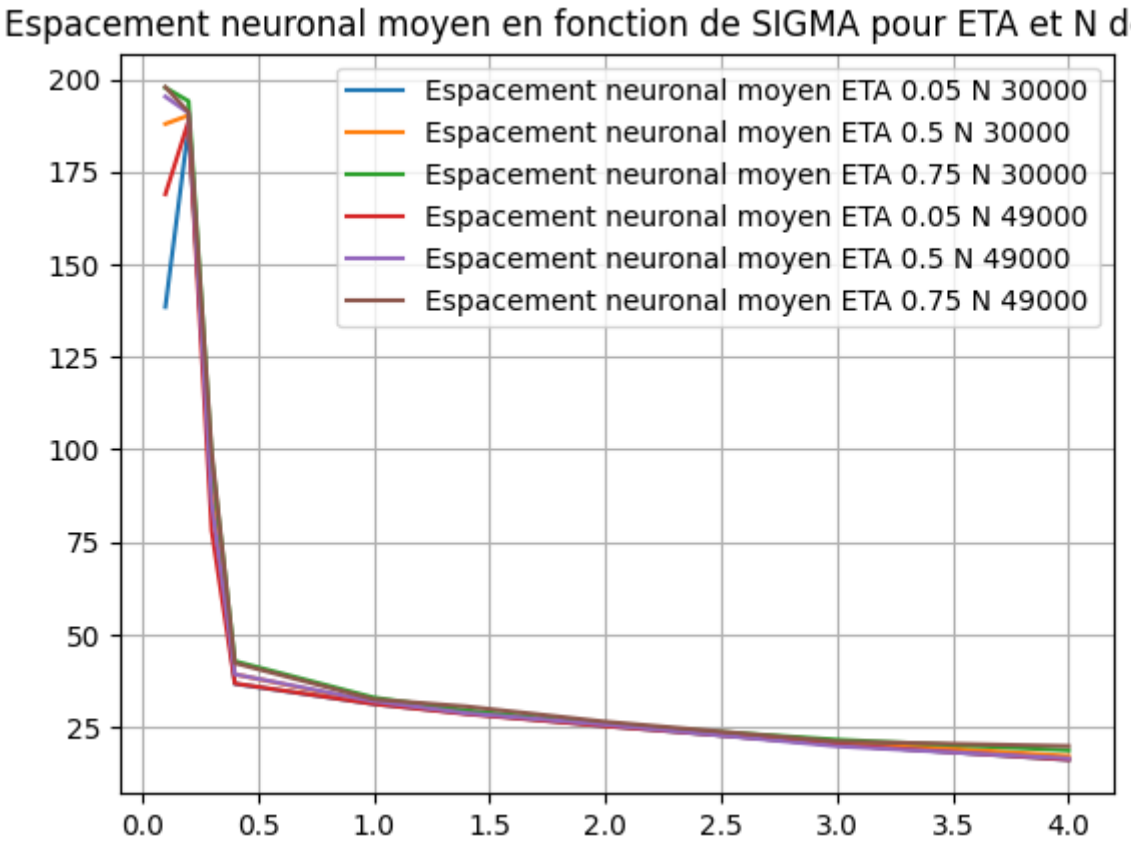

Poids dans l'espace d'entree
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

sigma = 8

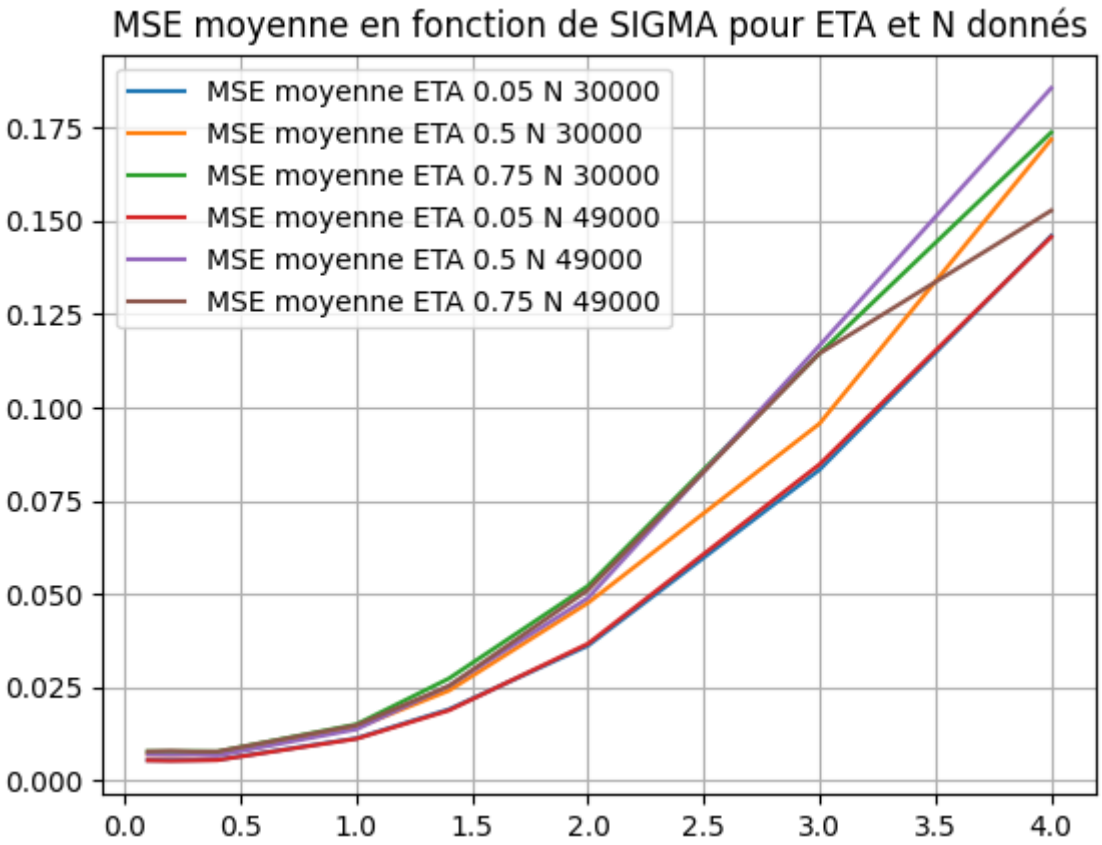


Nous voyons sur ces captures que la valeurs de σ à un impact sur l'espacement entre les points. Comme conjecturé dans la partie 3.2, la valeur de σ est inversement proportionnelle à l'espacement

entre les neuronnes. Cela signifie qu'au plus σ sera élevé, plus l'espacement neuronal sera faible :

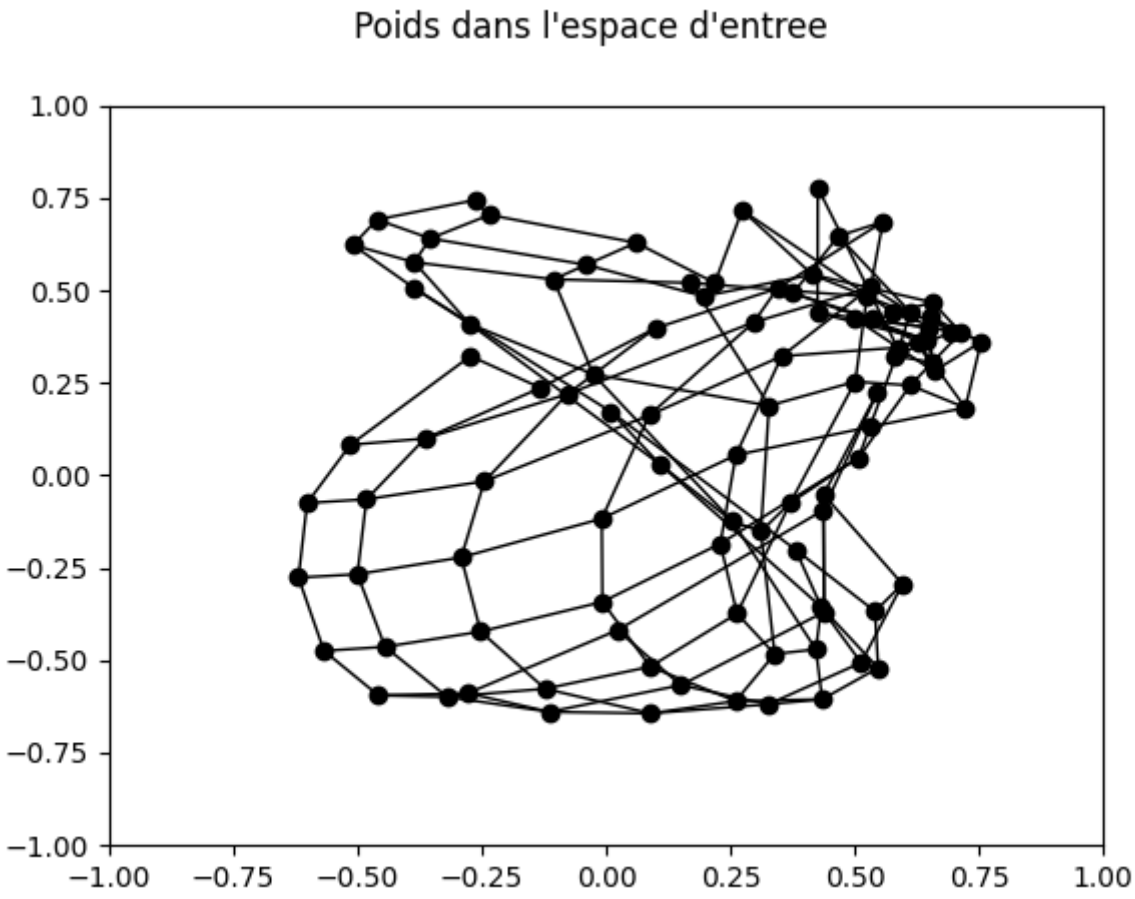


Cependant cela n'est pas le cas pour la MSE qui augmentera quand σ deviendra trop grand car les neurones seront trop concentrés pour occuper le jeu de données de façon efficace :

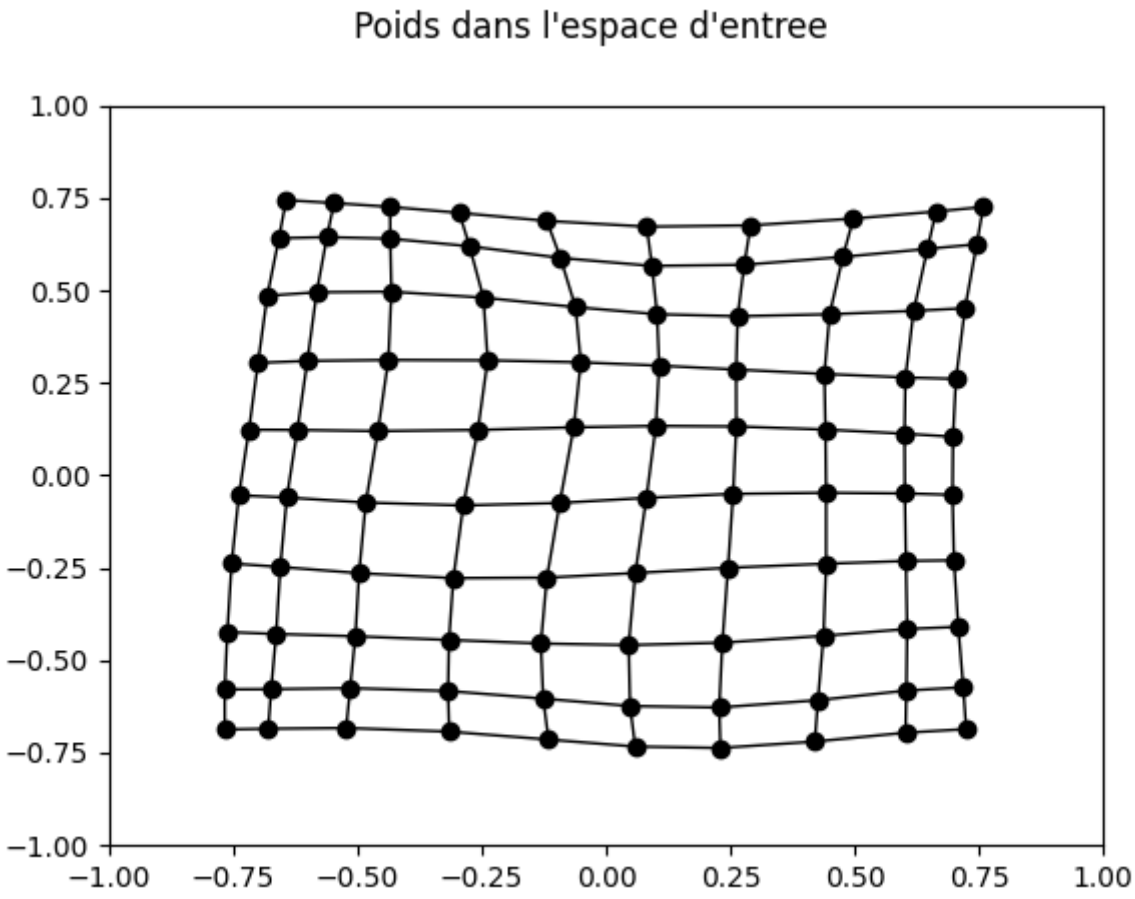


Variation de η

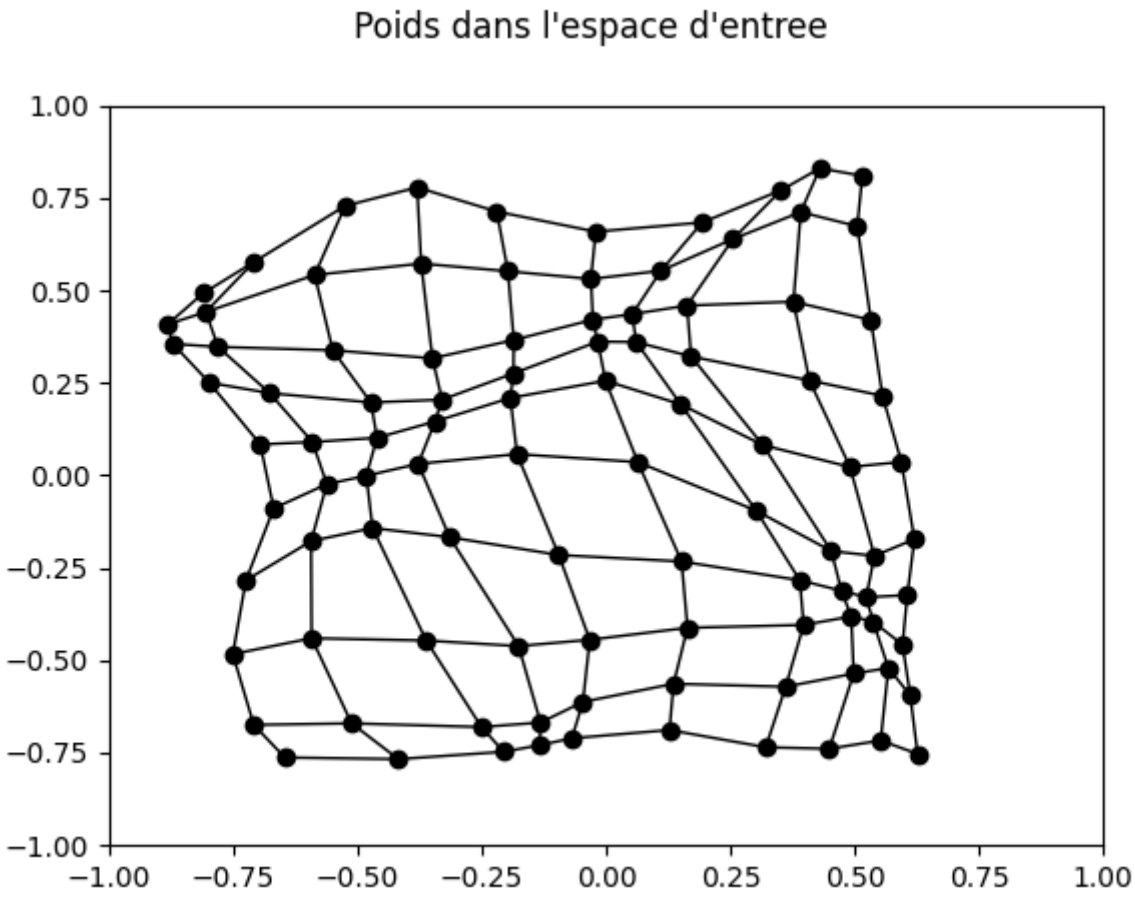
eta = 0.001


Poids dans l'espace d'entree
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

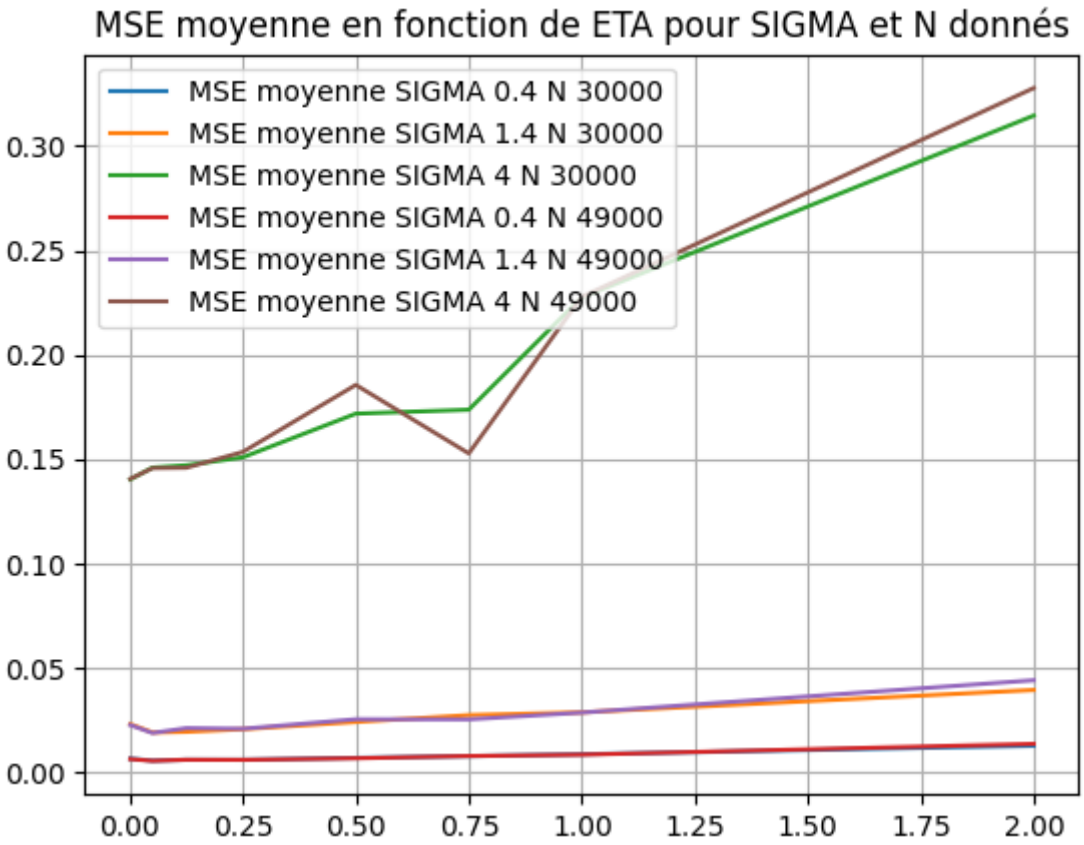
sigma = 0.05


Poids dans l'espace d'entree
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

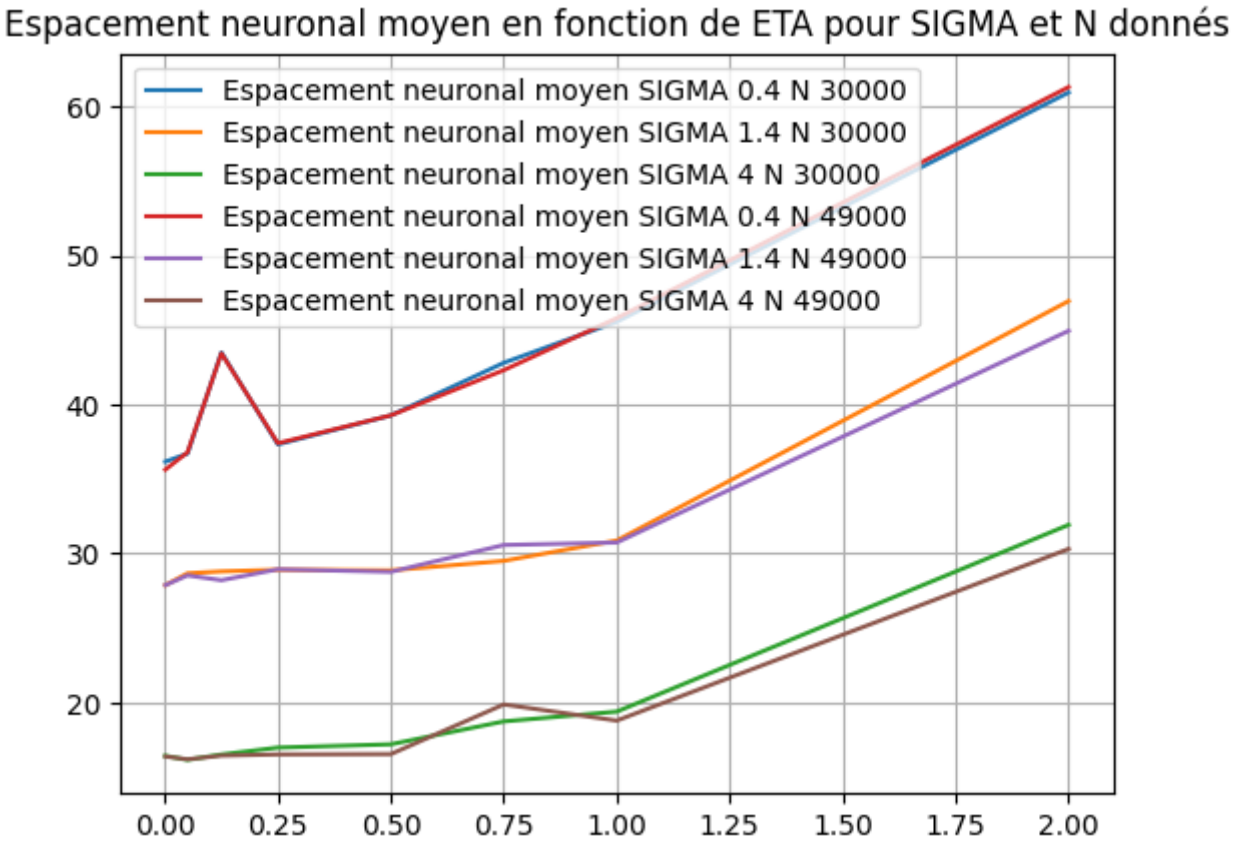
eta = 0.8



En étudiant ces captures, on peut confirmer qu'une valeur faible pour η donnera une carte brouillon car les neurones s'organiseront trop lentement. Cependant, on pourrait naturellement s'attendre à avoir de bon résultat avec un η haut mais comme conjecturé, cela n'est pas le cas car les poids vont évoluer trop brutalement et les neurones gagnants vont dépasser les données et vont avoir du mal à se stabiliser sur ces dernières et vont plutôt tourner autour, cela conduit inévitablement à une carte également brouillon. Il faut donc une valeur d'η qui soit assez réfléchi pour ne pas tomber dans l'un des deux extrêmes.
On peut, cependant, voir qu'une valeur faible pour η est préférable pour minimiser la MSE :

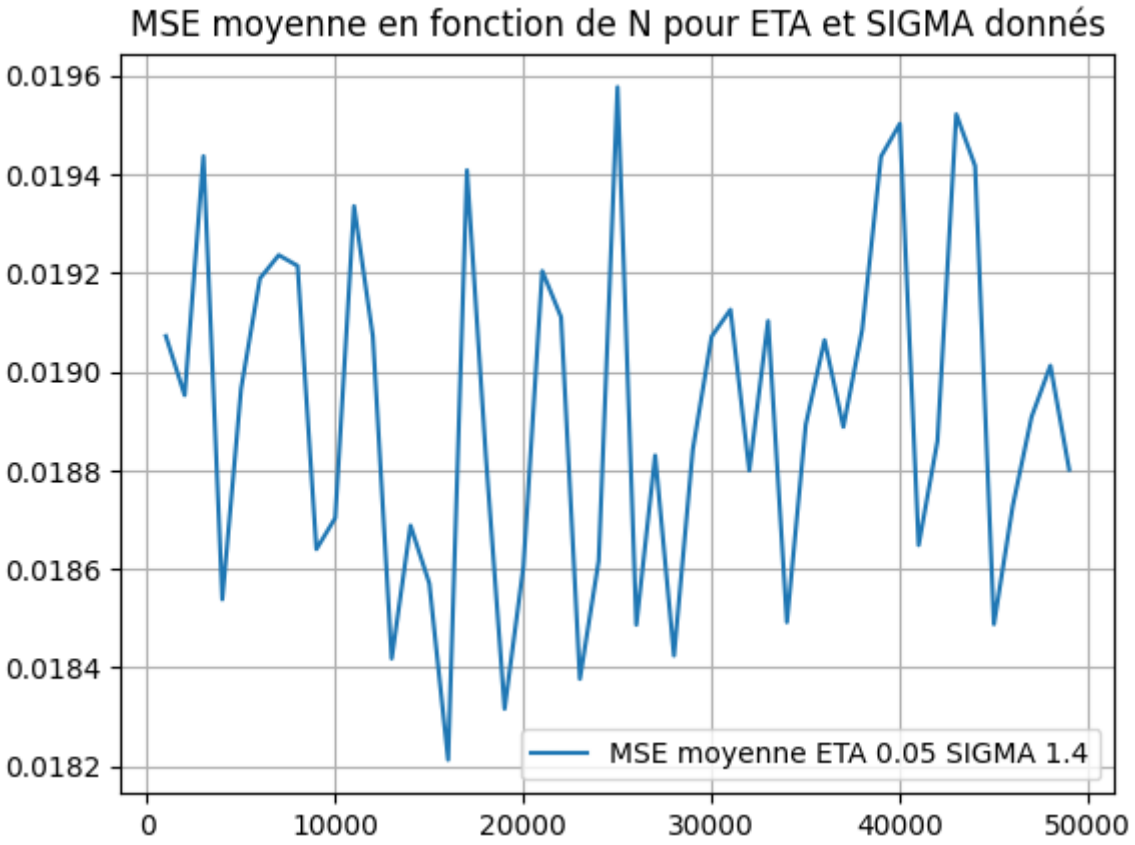


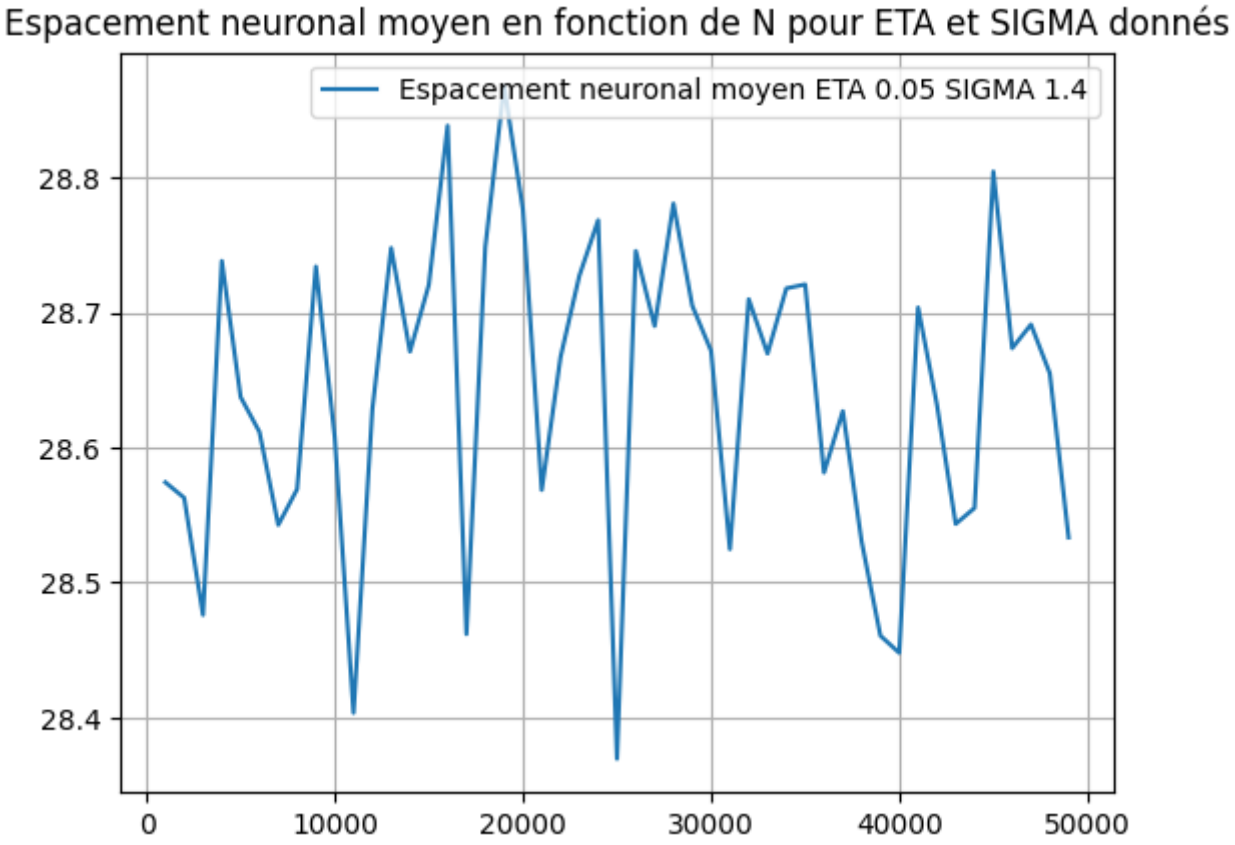
Et de façon similaire, l'espacement neuronal croit également en fonction d'η, ce qui n'est pas intuitif mais est dû au fait que les neurones s'éparpilleront plus facilement avec un haut taux d'apprentissage, et les neurones non-gagnants auront plus de mal à se rapprocher de ces derniers :



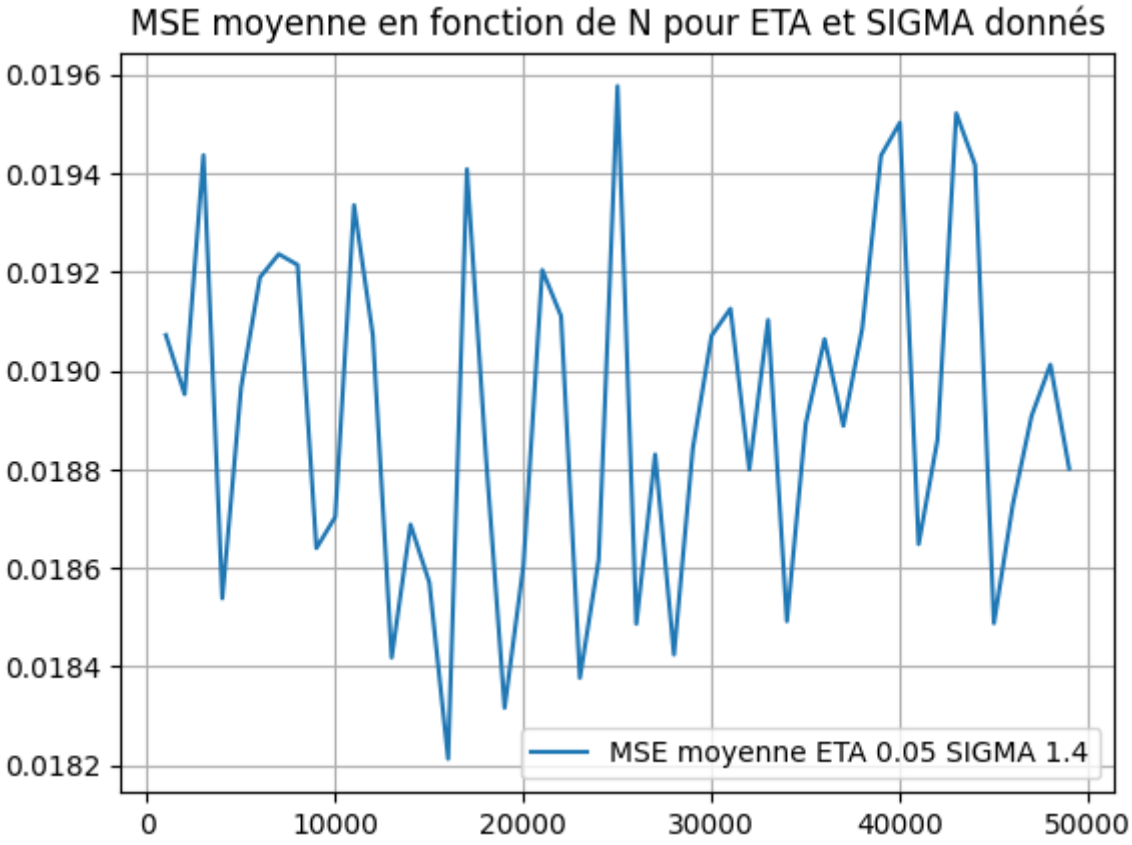
Variation de N

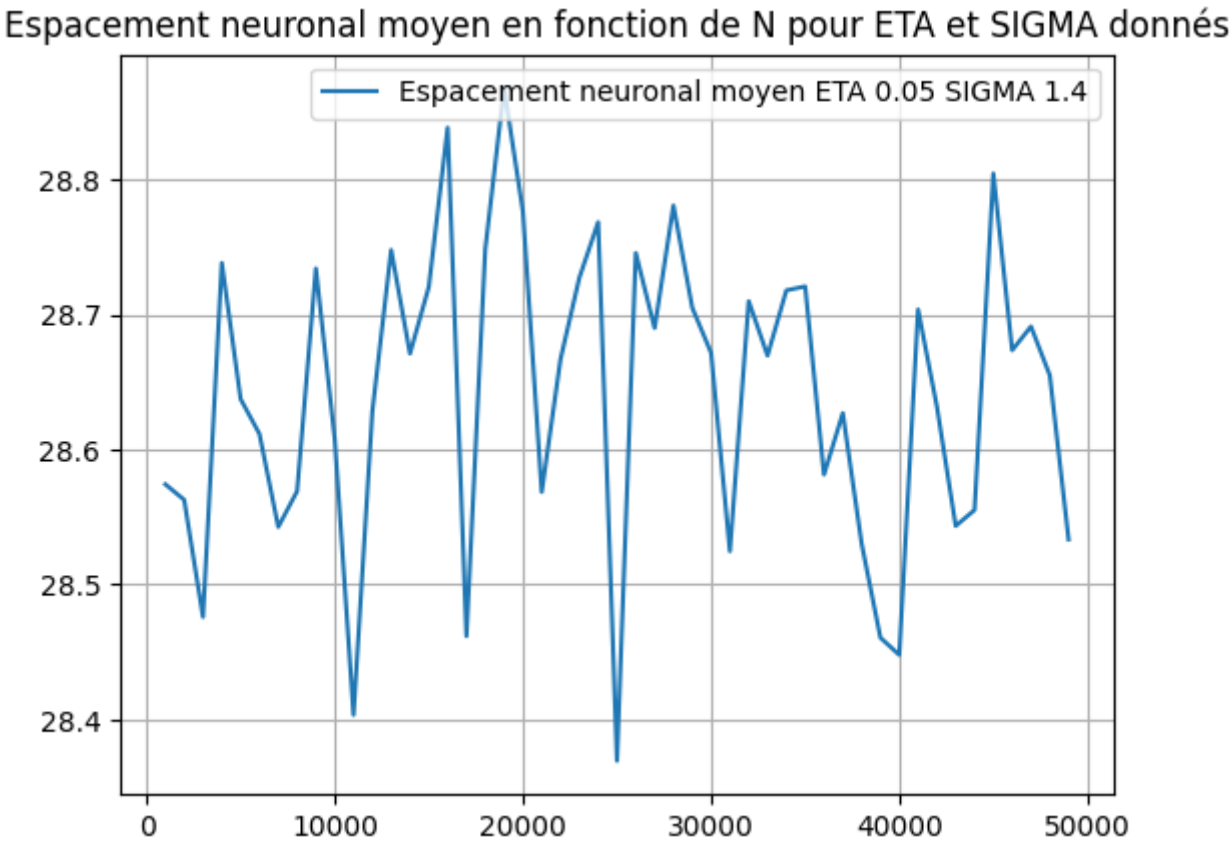
Il est logique de penser qu'un nombre élevé de pas laissera au réseau de neurones le temps d'atteindre un état "fixe" dans lequel peu de changement au niveau des poids surviendront à la prochaine itération. Cependant l'influence est moins flagrante que cela car il ne suffit que quelques milliers d'itérations avant que la carte se stabilise.





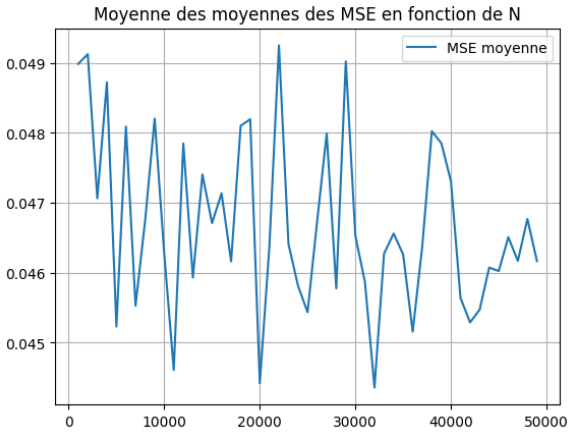
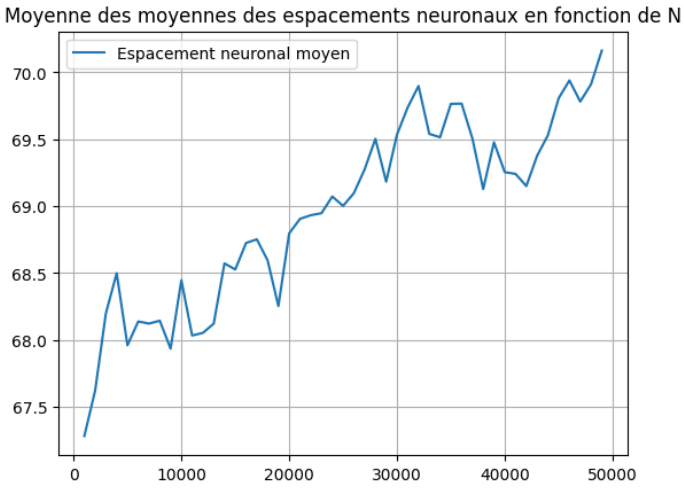
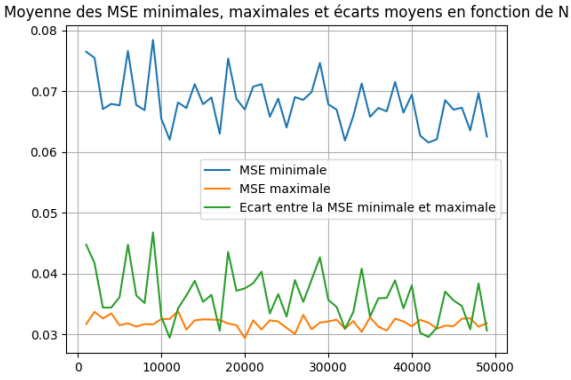
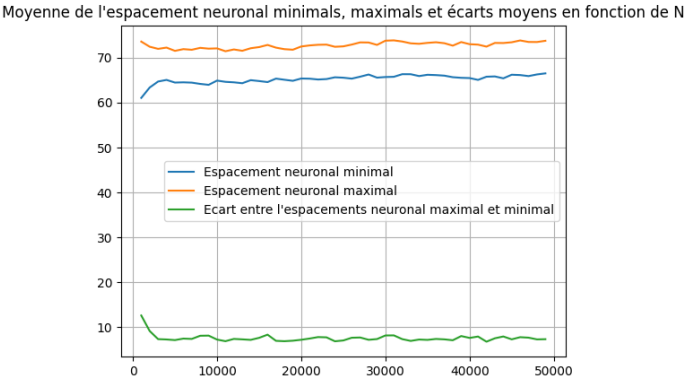
Les fluctuations de la MSE et de l'espacement neuronal sont faibles très rapidement et ne permettent pas de justifier un N très grand au lieu d'un N grand car quelques milliers de pas suffisent à obtenir un réseau stable.





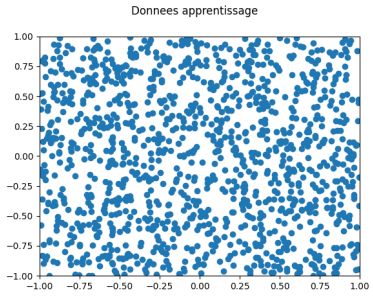
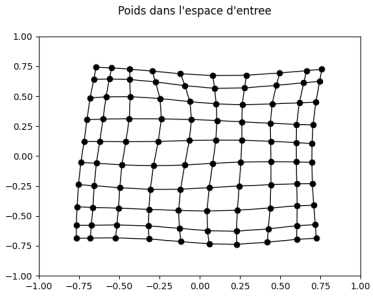
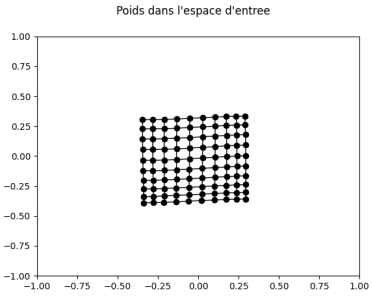
Ce propos concernent plutôt les couples η σ qui fonctionnent bien ensemble (comme par exemple 0.05 et 1.4 respectivement), mais dans le cas de couples moins synergiques, un N élevé peut se

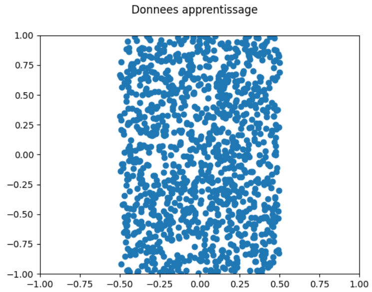
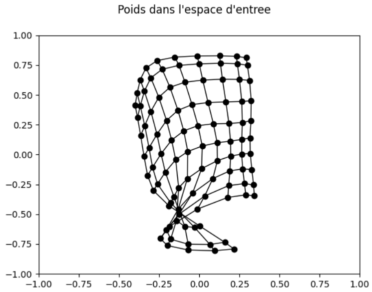
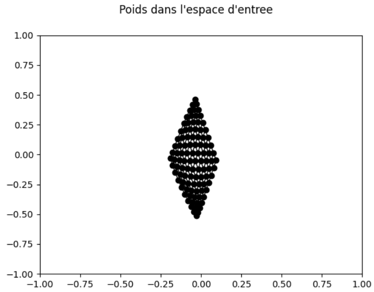
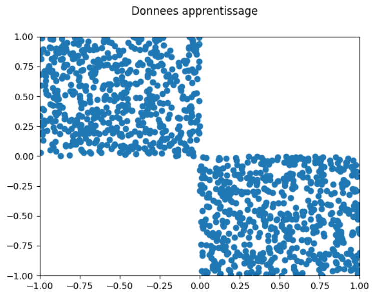
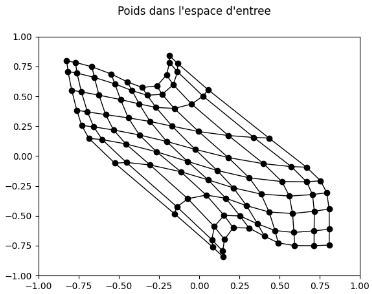
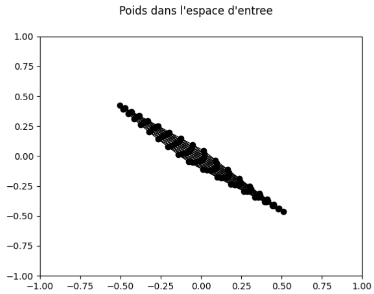
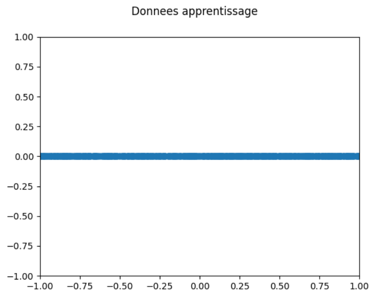
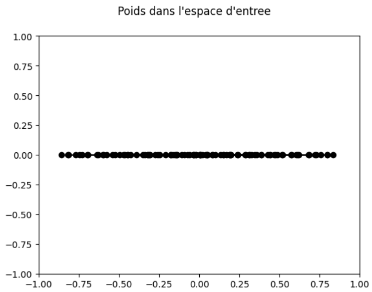
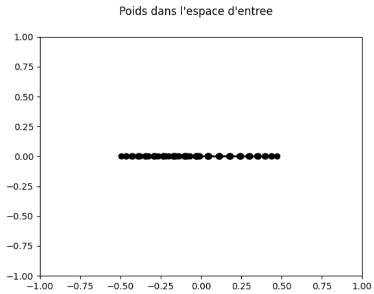
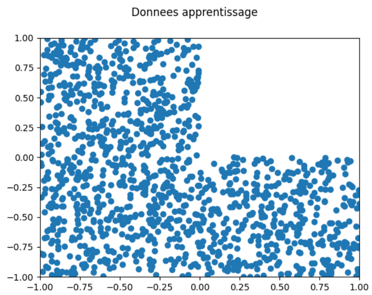
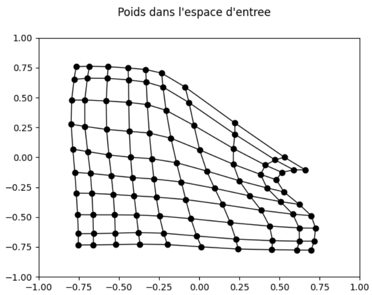
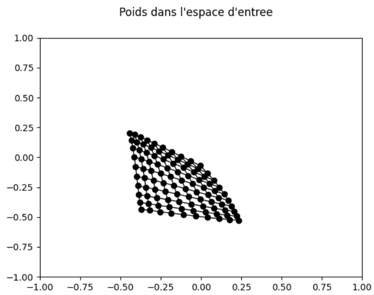
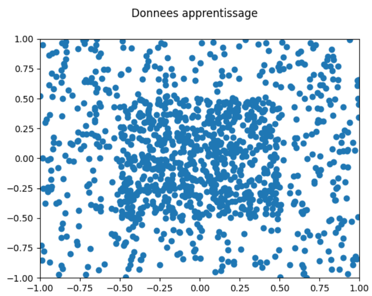
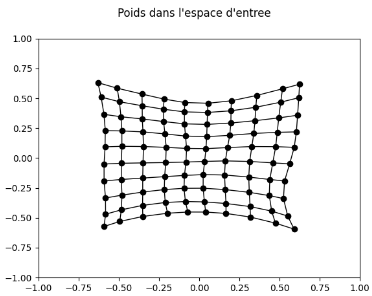
révélé intéressant comme par exemple si on prend la moyenne de la MSE et de l'espacement neuronal pour tous les couples étudiés :

MSE	Espacement neuronal
	
	

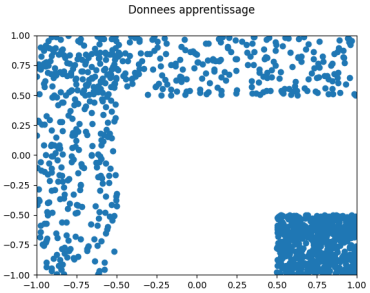
Modification de la carte et du jeu de données

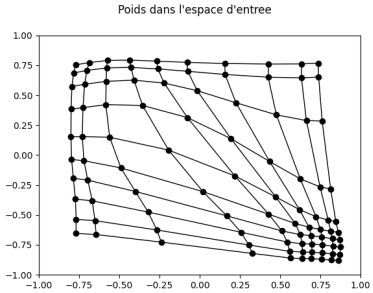
Conformément aux hypothèses faites dans la partie 3, si les données sont répartis non-uniformément ou dans des zones discontinues/disjointes, les neurones vont se répartir près des zones riches en données avec une minorité de neurones qui feront soit le pont au niveau des zones pauvres, soit qui les occuperont avec une densité en neurone proportionnelle à la densité des données si les zones ne sont pas uniformément distribuées :

Samples	$\eta=0.05, \sigma=1.4$	$\eta=0.05, \sigma=5$
		

Samples	$\eta=0.05, \sigma=1.4$	$\eta=0.05, \sigma=5$
		
		
		
		
		

Samples	$\eta=0.05, \sigma=1.4$	$\eta=0.05, \sigma=5$





4.4 Bras robotique

Pour cette partie, le code se trouve dans le fichier `bras_robotique.py`

Une fois la carte apprise, comment faire pour prédire la position qu'aura le bras étant donnée une position motrice ? Comment prédire la position motrice étant donnée une position spatiale que l'on souhaite atteindre ? Expliquer/justifer le principe et implémentez le.

Une fois la carte apprise, chaque neurone "stockera" dans son poids une partie de la résolution de l'équation pour une valeur donnée. Plus précisément, on retrouvera la correspondance entre les positions (x,y) de la main et les angles ($\theta1$; $\theta2$). Donc, pour un vecteur de position (x,y) , nous pourrons trouver un neurone qui a une valeur similaire/proche dans la première partie de son poids et, en regardant la deuxième partie du vecteur, nous trouverons la valeur ($\theta1$; $\theta2$) correspondante (et inversement). À moins d'avoir un neurone qui possède exactement le même (x,y), les angles que nous trouverons seront approchés. Pour être le plus précis possible, nous allons utiliser, en plus de ce neuronne, les autre neurones possèdant une valeur proche. Cette valeur sera inversement pondérée par rapport à la distance entre la position et le poid du neurone.

Voici le code correspondant à cette prédiction, ainsi que les calculs de vérifications :

```
def find_position(self, theta1, theta2):
    total_distance = 0
    neuron_distances: Dict[Neuron, float] = dict()
    for row in self.map :
      for neuron in row :
        distance = sqrt(pow(neuron.weights[0] - theta1, 2) + pow(neuron.weights[1]
- theta2, 2))
        neuron_distances[neuron] = distance
        total_distance += distance

    coeff_sum = 0
    position = [0,0]

    for neuron, distance in neuron_distances.items():
      ratio = total_distance/distance
      coeff_sum += ratio
      position[0] += neuron.weights[2]*ratio
      position[1] += neuron.weights[3]*ratio
```

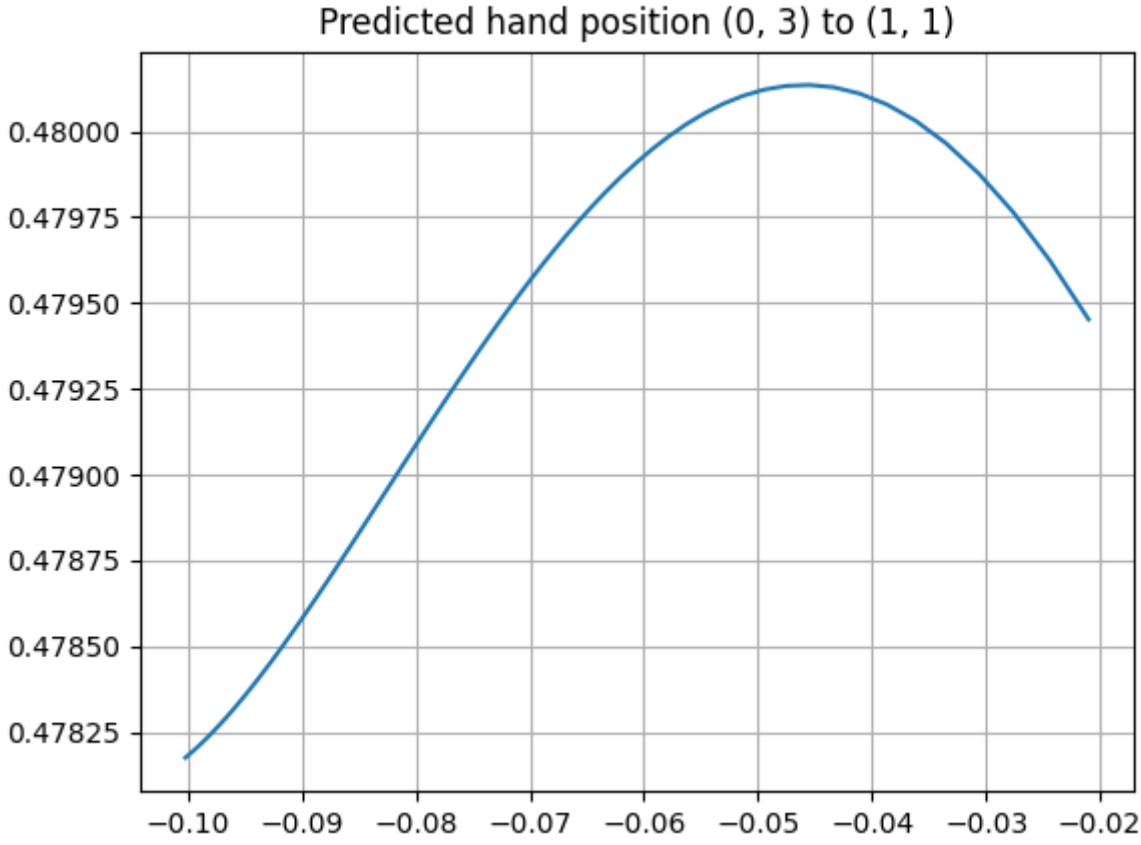
```
        position[0] /= coeff_sum
        position[1] /= coeff_sum

        return position

    t1 = 2
    t2 = 3

    print(network.find_position(t1,t2))

    x = l1*numpy.cos(t1)+l2*numpy.cos(t1+t2)
    y = l1*numpy.sin(t1)+l2*numpy.sin(t1+t2)

    print(f"x {x} y {y}")
```

On veut déplacer le bras d'une position motrice ($\theta1$;$\theta2$) à une nouvelle ($\theta1$; $\theta2$). En utilisant la carte apprise, comment prédire la suite des positions spatiales prise par la main ? On demande ici de pouvoir tracer grossièrement la trajectoire, pas forcément d'avoir la fonction exacte de toutes les positions prises. Expliquer/justifer le principe et implémentez le.

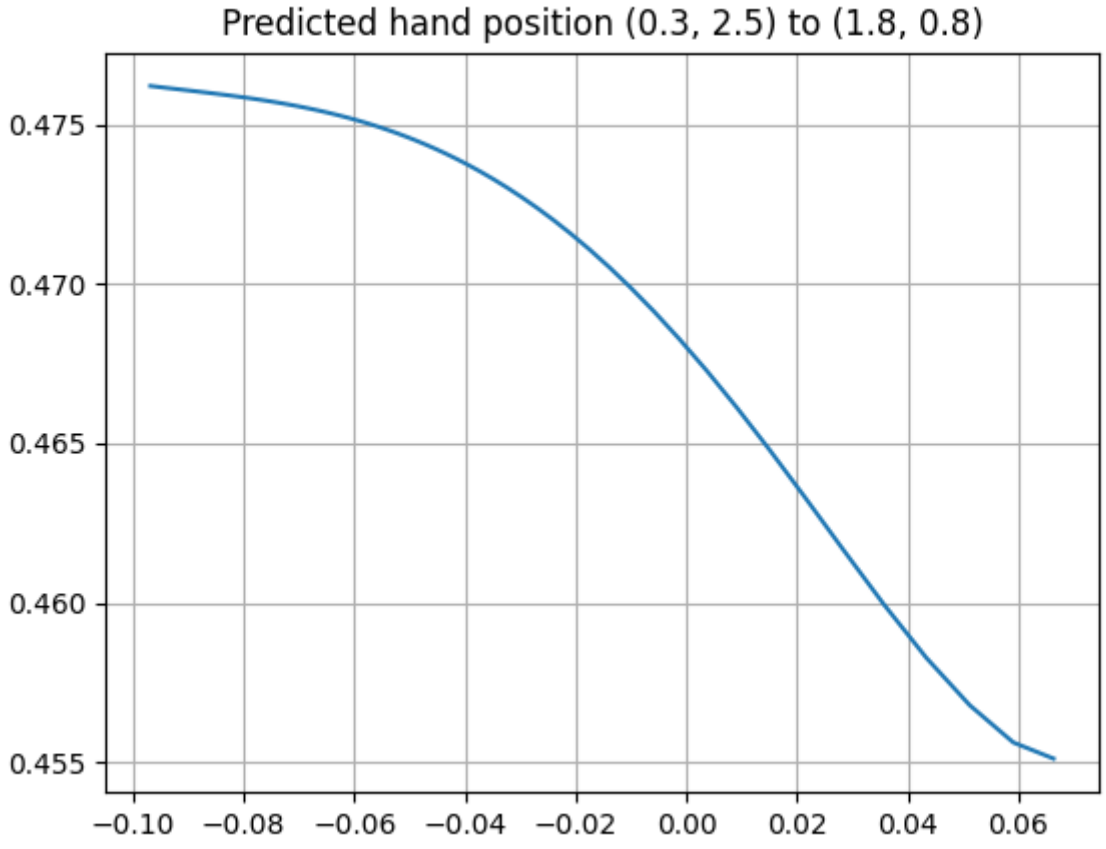
Pour prédire la suite des positions spatiales prisent par la main, il nous faut décomposer le mouvement en un certain nombre d'étapes. À chacune des ces étapes, nous allons effectuer la même action de prédiction que dans la question précédente. Cela nous donnera une courbe representant la position à chacune des étapes.

Afin de lisser la courbe, nous avons choisi de réaliser 100 étapes. Voici les graphiques correspondant à différents mouvement :

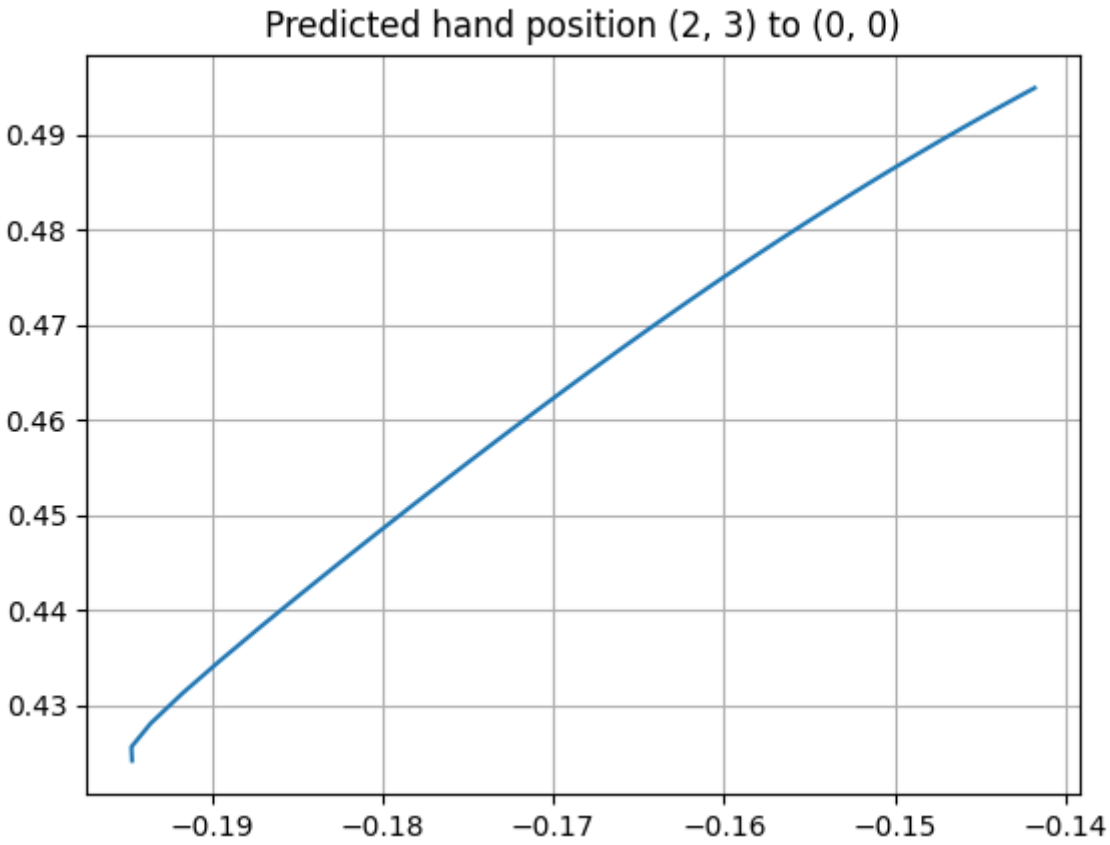
- Déplacement de la position (0,3) à (1,1)



- Déplacement de la position (0.3,2.5) à (1.8,0.8)



- Déplacement de la position (2,3) à (0,0)



Voici le code correspondant à ces générations :

```
t1p = 0
t2p = 0

step_number = 100

delta = sqrt(pow(t1 - t1p, 2) + pow(t2 - t2p, 2))

steps = list()

step_size = delta / step_number

for i in range(step_number + 1):
    stheta1 = t1 + (t1 - t1p) * step_size * i
    stheta2 = t2 + (t2 - t2p) * step_size * i
    steps.append(network.find_position(stheta1, stheta2))

plt.plot([step[0] for step in steps], [step[1] for step in steps])
plt.grid(True)
plt.title(f"Predicted hand position ({t1}, {t2}) to ({t1p}, {t2p})")
plt.show()
```