Rapport TP Réseaux de Neurones - Kohonen

👨‍🎓 Rémi MARTINEZ et Baptiste AUBERT

Rappel Equation : $\Delta w_{ij} = \eta e^{-\frac{||j-j^*||_c^2}{2\sigma^2}}(x_i - w_{ij})$

3. Etude 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 ?

  👉 Si on a $\eta = 0$, on aura : $\Delta w_{ij} = 0 * e^{-\frac{||j-j^*||_c^2}{2\sigma^2}}(x_i - w_{ij}) = 0$.
  Ainsi, la valeur des poids sera inchangée : car $\Delta w_{ij} = 0$, et ce Δ représente l'écart de poids entre une itération et une autre.

- Même question dans le cas ou η = 1.

  👉 Avec $\eta = 1$, on obtient :

$$\begin{aligned}\Delta w_{ij} &= 1 * e^{\frac{0}{2\sigma^2}}(x_i - w_{ij}) \\ &= 1 * e^{0}(x_i - w_{ij}) \\ &= (x_i - w_{ij})\end{aligned}$$

La valeur des poids sera la valeur actuelle des poids + x_i

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

  👉 Le poids modifié après sa variation Δw_{ij} sera entre W^* et X.

- Que va-t-il se passer si η > 1 ?

  👉 Avec $\eta > 1$, le poids modifié va dépasser X ce qui va donner un apprentissage incorrect : le vecteur d’entrée sera sur-représenté et l’ancien vecteur correspondant au neurone gagnat sera sous-représenté.

3.2 Influence de σ

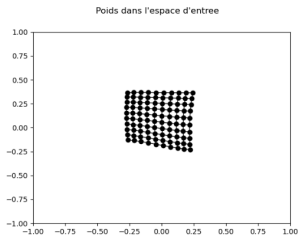
- Si σ augmente, les neurones proches du neurone gagnant (dans la carte), vont-ils plus ou moins apprendre l’entrée courante ?

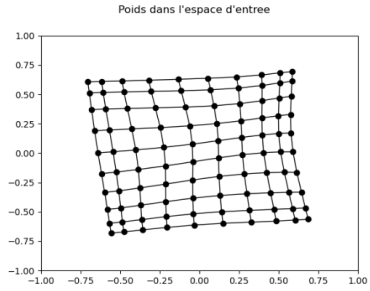
  👉 Lorsque σ augmente, le dénominateur $2\sigma^2$ augmente, donc la fraction $-\frac{||j-j^*||_c^2}{2\sigma^2}$ va tendre vers 0. Comme on va tendre vers $e^0 = 1$ (qui représente le maximum pour une exponentielle négative), les neurones vont apprendre davantage.

- 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” ?

  👉 Plus σ est grand, à convergence, plus l’auto-organisation obtenue sera resserrée.

$\sigma = 4.0$ $\sigma = 1.4$





- 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 ?

  👉 On cherche une méthode qui va retourner un chiffre dont la valeur sera élevée si carte est "lâche" ou faible si "resserrée"

  On va d'abord calculer la moyenne des poids pour chaque neurone, puis faire cette moyenne - le poids de chaque neurone ⇒ on obtient une distance d'un neurone avec la moyenne

  Rappel pour les poids $\Delta w_{ij} = \eta e^{-\frac{||j-j^*||_c^2}{2\sigma^2}}(x_i - w_{ij})$ ⇒ Donc si σ est plus grand, le résultat de la fraction en exposant sera plus faible et ainsi l'écart entre les neurones sera réduit (on rappelle que l'exponentielle tendra vers 1).

3.3 Influence de la distribution d'entrée

Prenons le cas très simple d'une carte a 1 neurone qui reçoit deux entrées X_1 et X_2

- Si X_1 et X_2 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) ?

👉 X_1 → le poids du neurone va tendre vers X_1

X_2 → le poids du neurone va tendre vers X_2

A convergence → le poids du neurone va converger entre X_1 et X_2

$W^* = \frac{X_1+X_2}{2}$

- Même question si X_1 est présenté n fois plus que X_2

Le poids du neurone va tendre n fois plus vers X_1 que vers X_2.

On obtient:

$W^* = \frac{nX_1+X_2}{n+1}$

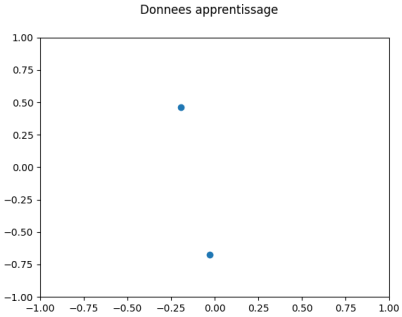
On avait précédemment 2 en dénominateur car on avait deux données en entrée.

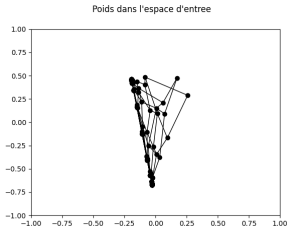
On a maintenant $n+1$ données en entrée.
Exemple : Si X_1 est $n = 2$ fois plus représenté que X_2, on aura 3 points.

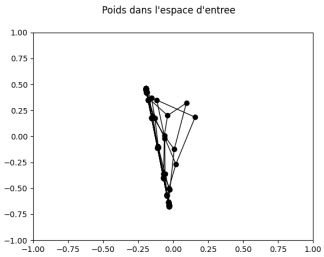
- En déduire comment vont se répartir les neurones en fonction de la densité des données

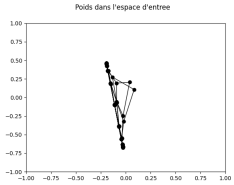
Exemple : 2 données en entrée

Les poids des neurones convergent entre X_1 et X_2

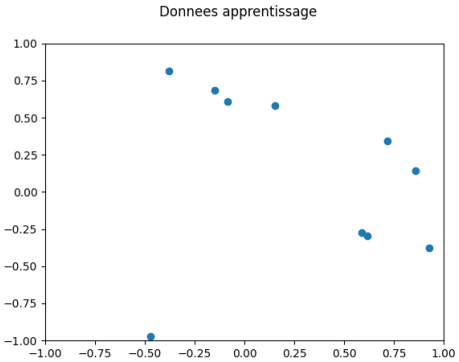


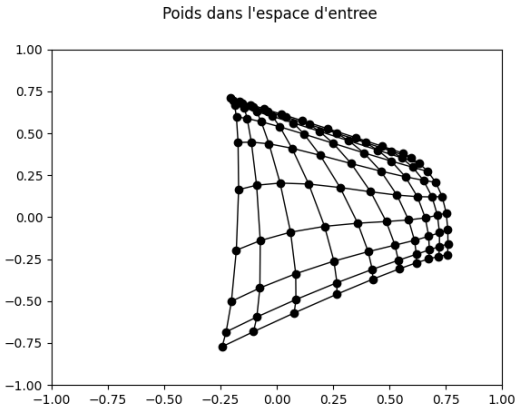






Exemple : 10 données en entrée





Les neurones vont se répartir selon la position des données d'apprentissages en entrée. Pour raisonner avec la quantification vectorielle, on associe un vecteur référent dans l'espace des données pour chaque observation, ce qui associe pour chaque neurone une position particulière sur la carte. Les vecteurs poids des neurones vont alors converger vers ces vecteurs référents d'entrée. On remarque d'ailleurs que les neurones sont semblables à la densité de la base d'apprentissage.

4. Etude pratique

4.2 Implémentation

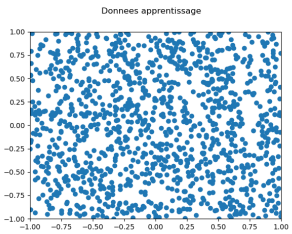
```python
def compute(self, x):
    """
@summary: Affecte à y la valeur de sortie du neurone (i.e. la distance entre son poids et l'entrée)
@param x: entrée du neurone
@type x: numpy array
"""
    self.y = numpy.linalg.norm(x - self.weights)

def learn(self, eta, sigma, posxbmu, posybmu, x):
    """
@summary: Modifie les poids selon la règle de Kohonen
@param eta: taux d'apprentissage
@type eta: float
@param sigma: largeur du voisinage
@type sigma: float
@param posxbmu: position en x du neurone gagnant (i.e. celui dont le poids est le plus proche de l'entrée)
@type posxbmu: int
@param posybmu: position en y du neurone gagnant (i.e. celui dont le poids est le plus proche de l'entrée)
@type posybmu: int
@param x: entrée du neurone
@type x: numpy array
"""
    self.weights[:] += eta * numpy.exp(
        -(numpy.power(numpy.abs(numpy.sqrt(numpy.power(posxbmu - self.posx, 2) + numpy.power(posybmu - self.posy, 2))), 2)
          / numpy.power(2 * sigma, 2))) * (x - self.weights)
```

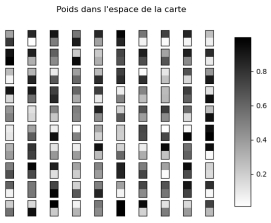
4.3 Analyse de l'algorithme

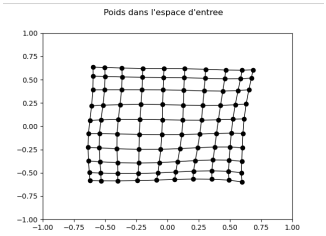
Dans cette section nous allons étudier l'influence de certains éléments sur le fonctionnement de l'algorithme de Kohonen en les modifiants à tour de rôle.

Données d'origines

η	σ	N	Taille de la carte	Forme de la carte	Dataset	Erreur de quantification vectorielle	Mo mo or
0.05	1.4	30000	10x10	Carrée	Ensemble de données 1	0.03618734733223264	-



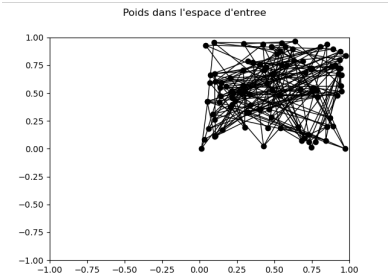


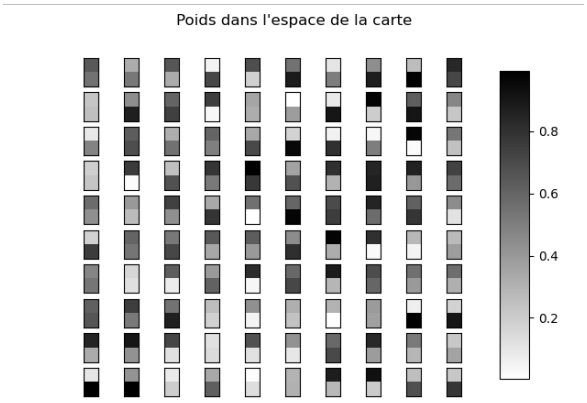


Taux d'apprentissage η

- $\eta = 0$

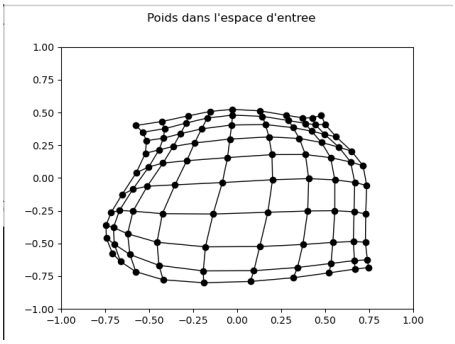
η	σ	N	Taille de la carte	Forme de la carte	Dataset	Erreur de quantification vectorielle	Mo mo or
0	1.4	30000	10x10	Carrée	Ensemble de données 1	0.36448555982277203	81

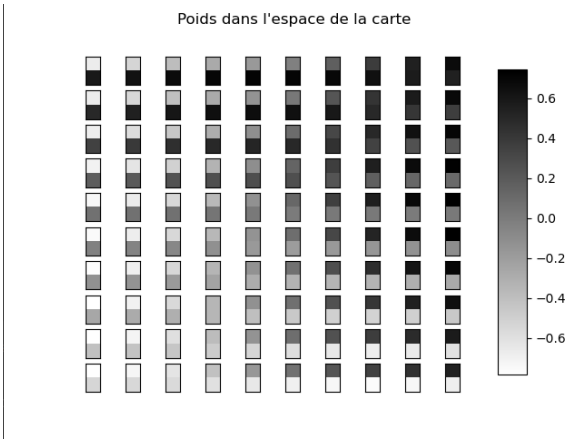




- $\eta = 0.5$

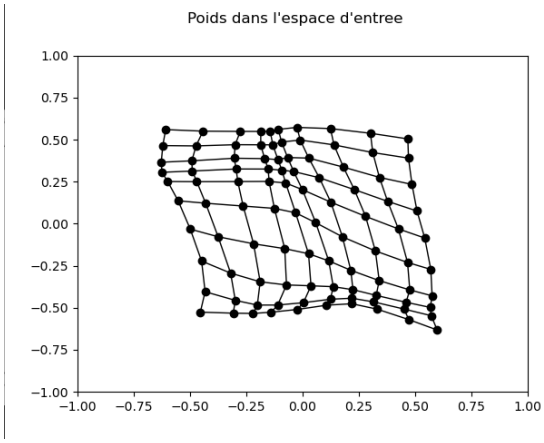
η	σ	N	Taille de la carte	Forme de la carte	Dataset	Erreur de quantification vectorielle	M n o
0.5	1.4	30000	10x10	Carrée	Ensemble de données 1	0.029655090499320427	5

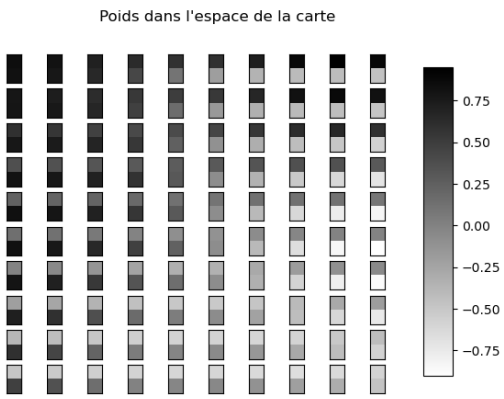




- $\eta = 0.75$

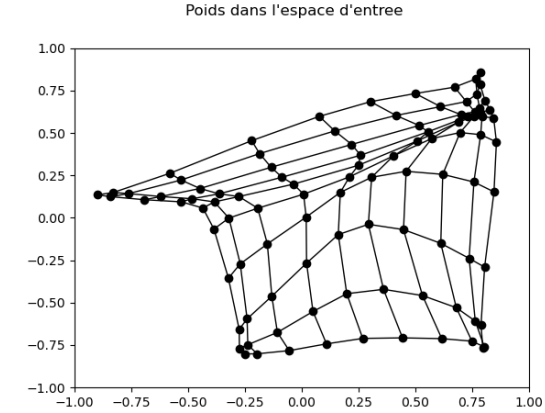
η	σ	N	Taille de la carte	Forme de la carte	Dataset	Erreur de quantification vectorielle	M m or
0.75	1.4	30000	10x10	Carrée	Ensemble de données 1	0.06044797760426964	46

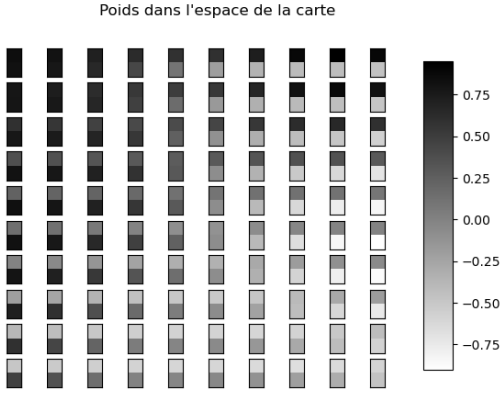




- $\eta = 0.99$

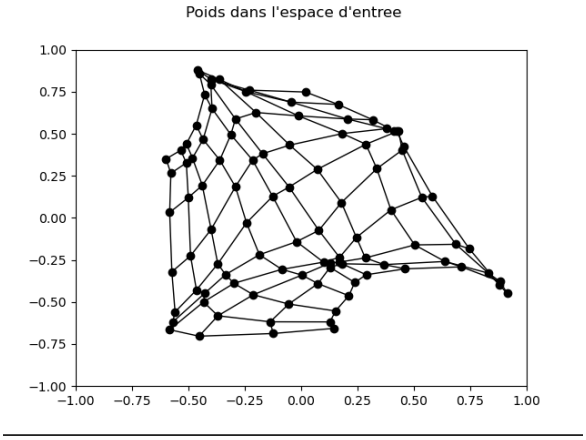
η	σ	N	Taille de la carte	Forme de la carte	Dataset	Erreur de quantification vectorielle	Mo me org
0.99	1.4	30000	10x10	Carrée	Ensemble de données 1	0.04251157743739089	47

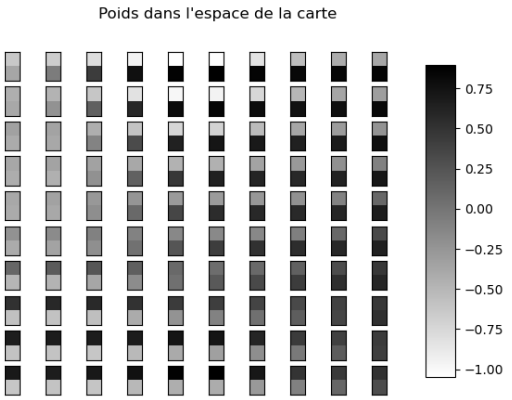




- $\eta = 1.25$

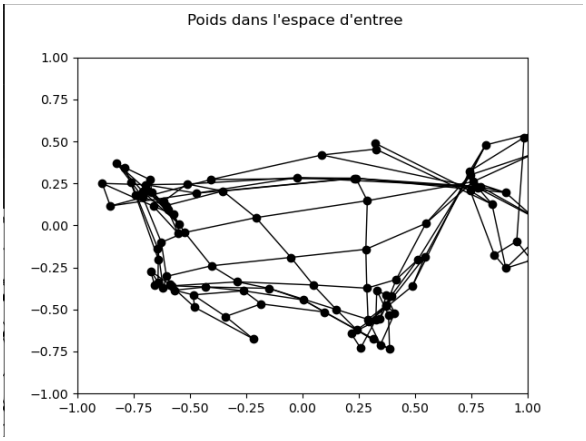
η	σ	N	Taille de la carte	Forme de la carte	Dataset	Erreur de quantification vectorielle	M n o
1.25	1.4	30000	10x10	Carrée	Ensemble de données 1	0.051817125215801695	4

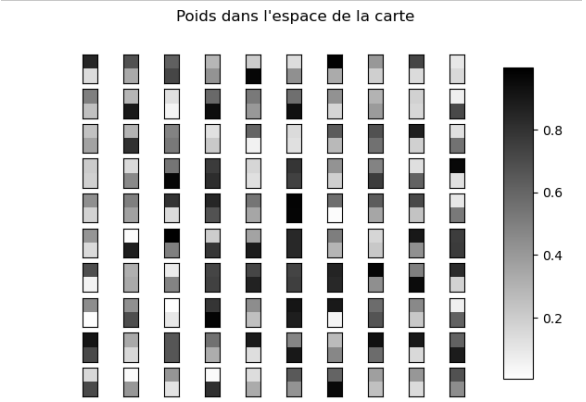




- $\eta = 2$

η	σ	N	Taille de la carte	Forme de la carte	Dataset	Erreur de quantification vectorielle	Mo me or
2	1.4	30000	10x10	Carrée	Ensemble de données 1	0.11544813655759029	55





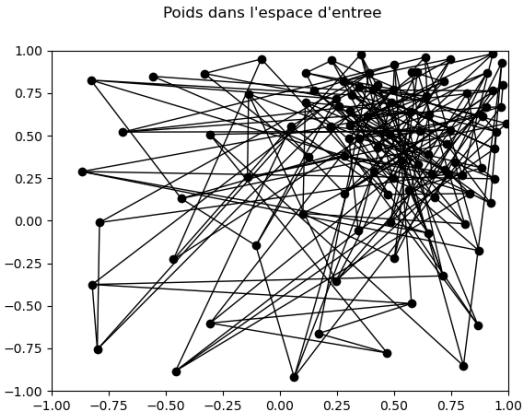
Remarque :
Lorsque $\eta = 0$, il n'y a pas d'apprentissage ⇒ les neurones n'apprennent rien et lorsque $\eta \to 0$ cela permet une auto-organisation plus stable.

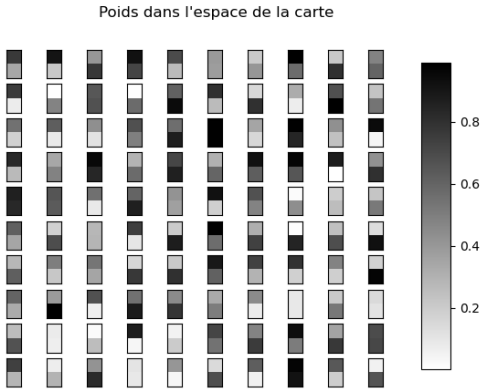
Inversement, si η est grand (≥ 1), on remarque une instabilité de l'organisation. On remarque que sur le graphique cela se traduit par des zones qui se regroupent et déforment la carte globale.

Largeur du voisinage σ

- $\sigma = 0.1$

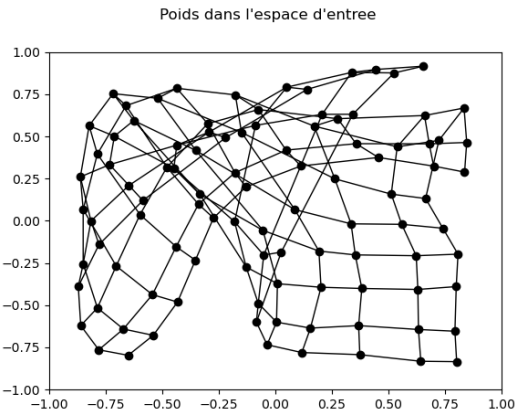
η	σ	N	Taille de la carte	Forme de la carte	Dataset	Erreur de quantification vectorielle	M n o
0.05	0.1	30000	10x10	Carrée	Ensemble de données 1	0.014072659845587822	8

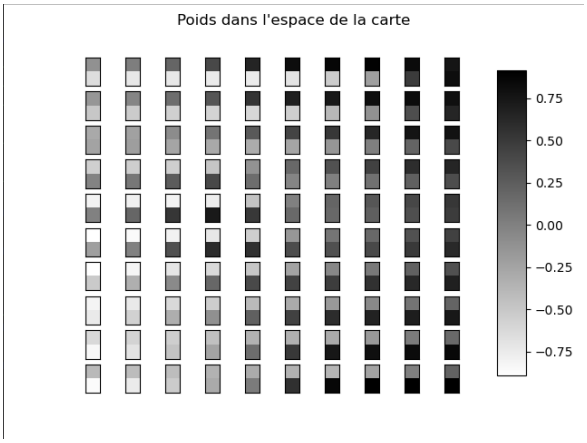




- $\sigma = 0.5$

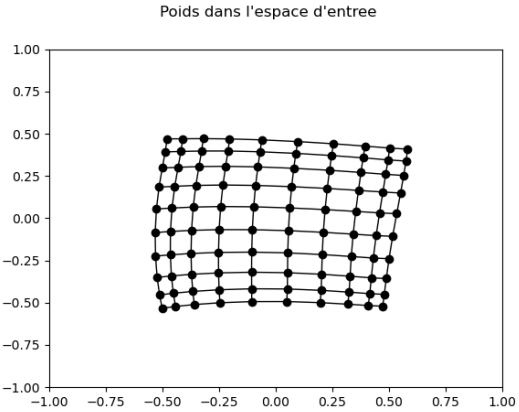
η	σ	N	Taille de la carte	Forme de la carte	Dataset	Erreur de quantification vectorielle	Me mo or
0.05	0.5	30000	10x10	Carrée	Ensemble de données 1	0.01075146857353243	57

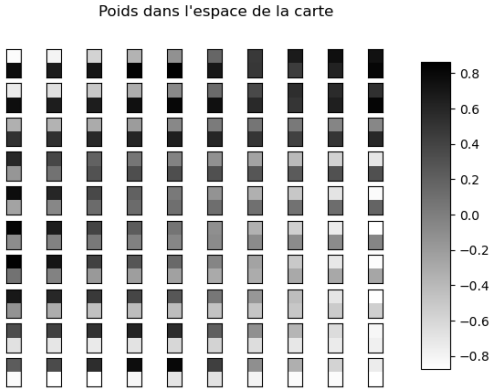




- $\sigma = 2$

η	σ	N	Taille de la carte	Forme de la carte	Dataset	Erreur de quantification vectorielle	Me mo or
0.05	2	30000	10x10	Carrée	Ensemble de données 1	0.07356578128504035	39



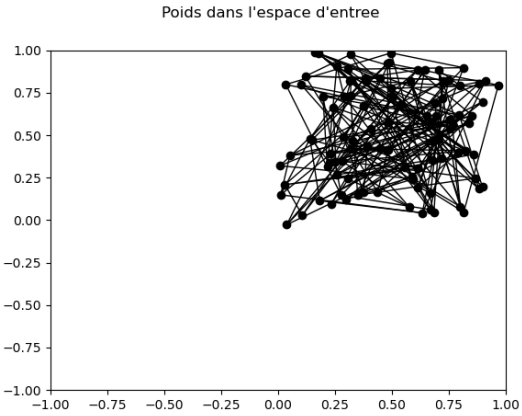


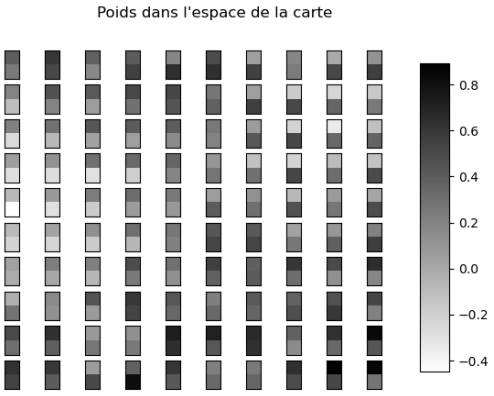
Remarque:
Lorsque $\sigma \to 0$, les neurones sont répartis de manière totalement désorganisée. Par contre, plus σ est grand, plus la répartition des neurones est resserée.

Nombre de pas de temps d'apprentissage N

- $N = 100$

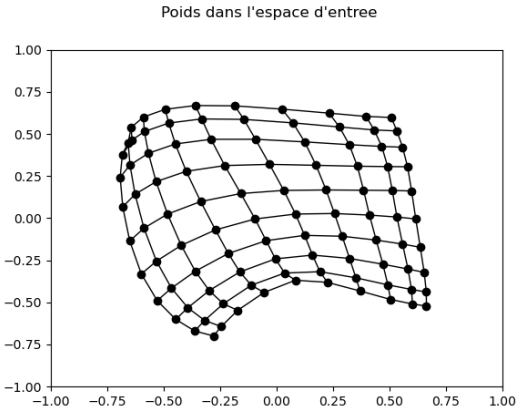
η	σ	N	Taille de la carte	Forme de la carte	Dataset	Erreur de quantification vectorielle	Me mo or
0.05	1.4	100	10x10	Carrée	Ensemble de données 1	0.14496492889272933	63

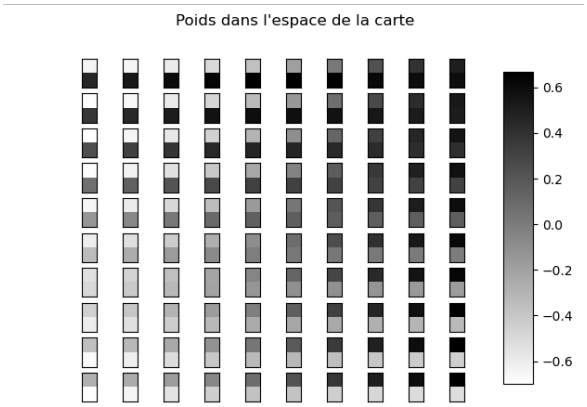




- $N = 1000$

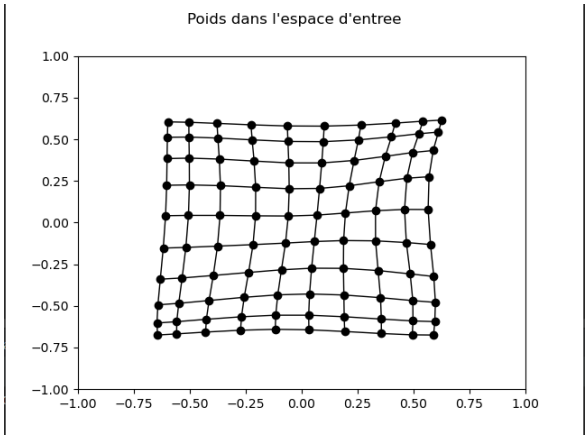
η	σ	N	Taille de la carte	Forme de la carte	Dataset	Erreur de quantification vectorielle	Me mo or
0.05	1.4	1000	10x10	Carrée	Ensemble de données 1	0.05039389257132509	47

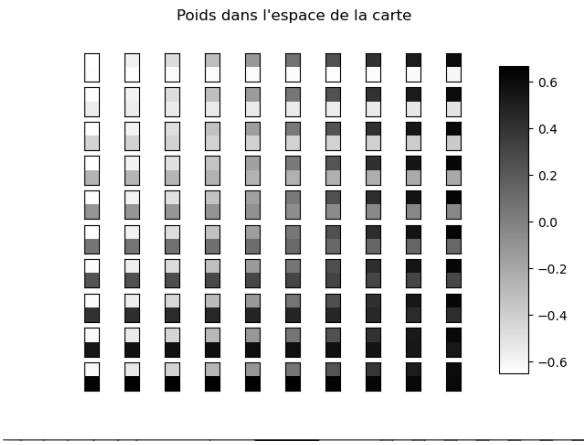




- $N = 10000$

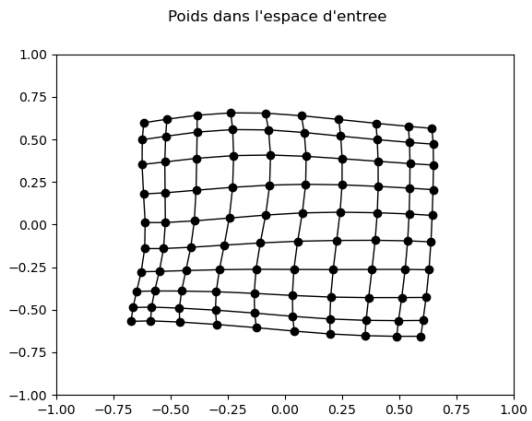
η	σ	N	Taille de la carte	Forme de la carte	Dataset	Erreur de quantification vectorielle	Mo me or
0.05	1.4	10000	10x10	Carrée	Ensemble de données 1	0.03467213310649545	45

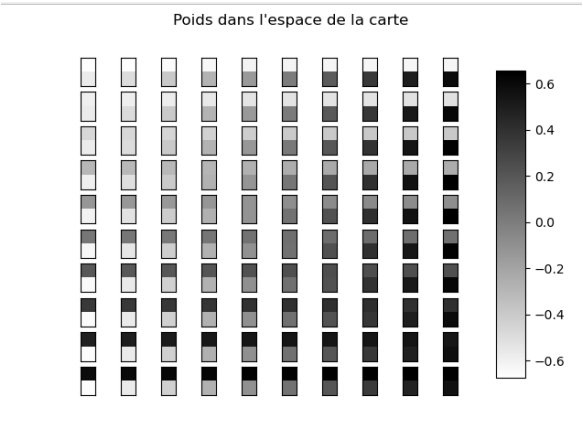




- $N = 100000$

η	σ	N	Taille de la carte	Forme de la carte	Dataset	Erreur de quantification vectorielle	Moye mesu orga
0.05	1.4	100000	10x10	Carrée	Ensemble de données 1	0.035890977910502	449.8



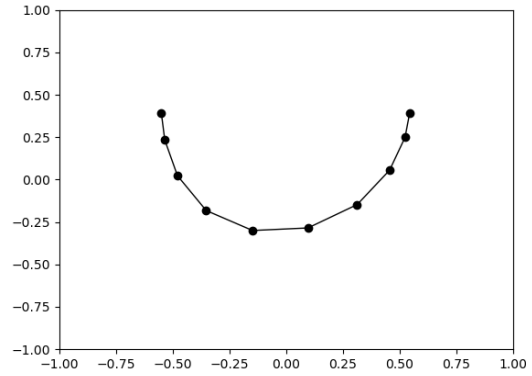


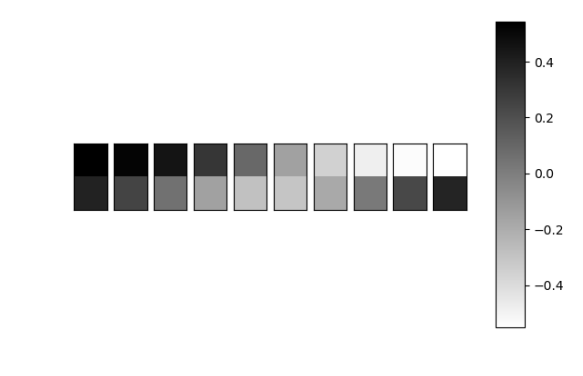
Remarque:
Lorsque $N \to 0$, le temps d'apprentissage est raccourci, donc les neurones ont moins le temps d'apprendre, ce qui donne des erreurs de quantification vectorielle plus élevées.
A contrario, si N est grand, alors le temps d'apprentissage va être plus long et va permettre aux neurones de mieux apprendre et par conséquent obtenir une erreur de quantification vectorielle plus élevée.

Taille et forme de la carte

- Ligne horizontale de taille 1x10

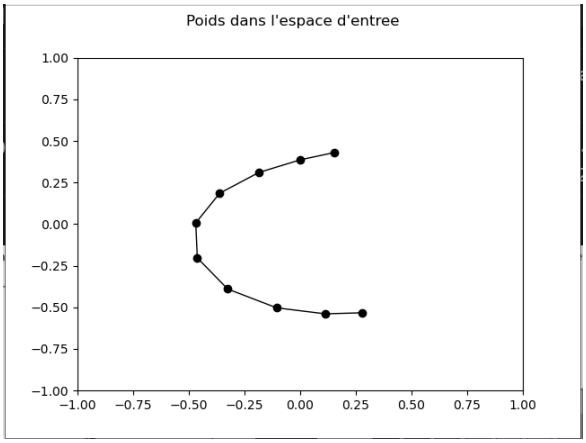
η	σ	N	Taille de la carte	Forme de la carte	Dataset	Erreur de quantification vectorielle	Mo me or
0.05	1.4	30000	1x10	Ligne horizontale	Ensemble de données 1	0.23266477748922307	38

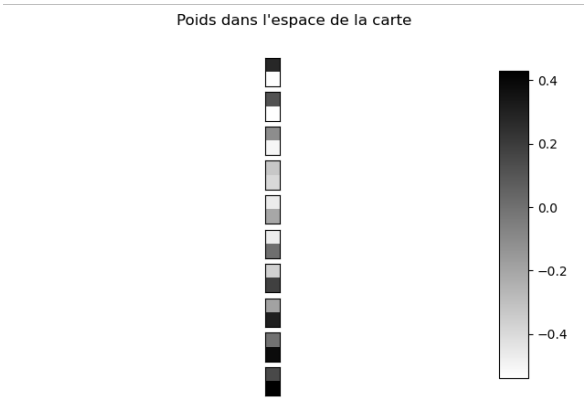




- Ligne verticale de taille 10x1

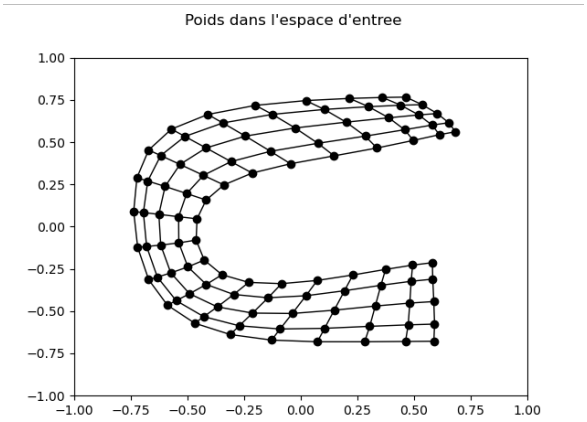
η	σ	N	Taille de la carte	Forme de la carte	Dataset	Erreur de quantification vectorielle	Moye mesu orga
0.05	1.4	30000	10x1	Ligne verticale	Ensemble de données 1	0.216097208671686	30.4(

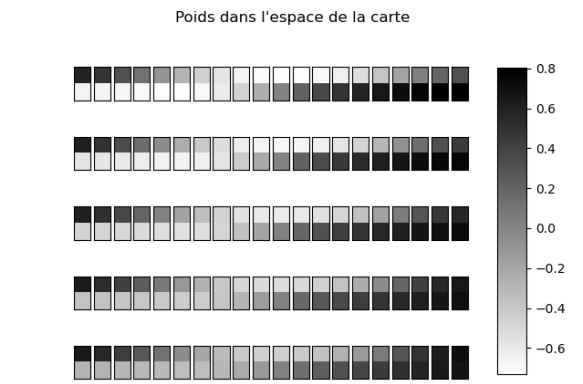




- Rectangle de taille 5x20

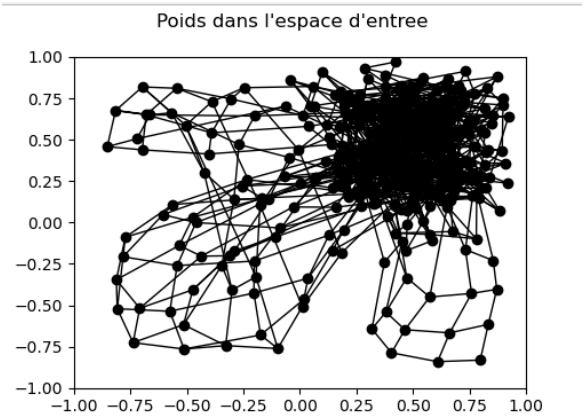
η	σ	N	Taille de la carte	Forme de la carte	Dataset	Erreur de quantification vectorielle	M n o
0.05	1.4	30000	5x20	Rectangle	Ensemble de données 1	0.010823620373695685	1

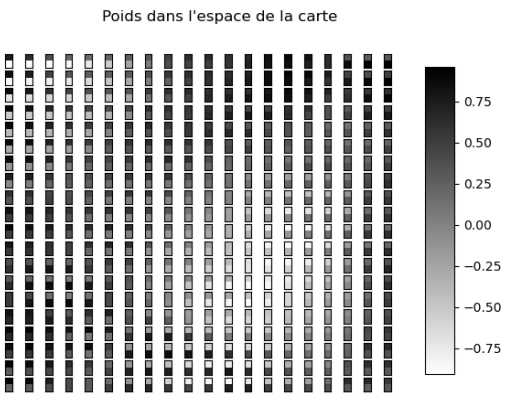




- Carré de grande taille 20x20

η	σ	N	Taille de la carte	Forme de la carte	Dataset	Erreur de quantification vectorielle	M n o
0.05	1.4	30000	20x20	Grand carré	Ensemble de données 1	0.004670712961246372	5



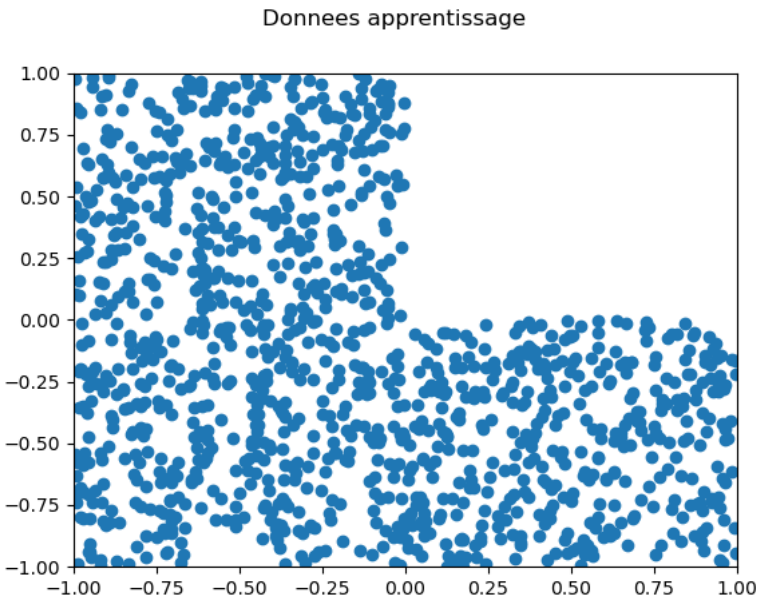


Remarque:
On remarque qu'une carte de petite taille s'organise plus vite et plus facilement qu'une carte de grande taille.

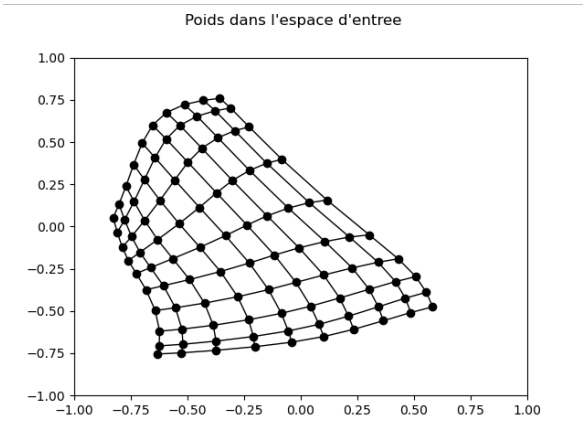
Influence du jeu de données

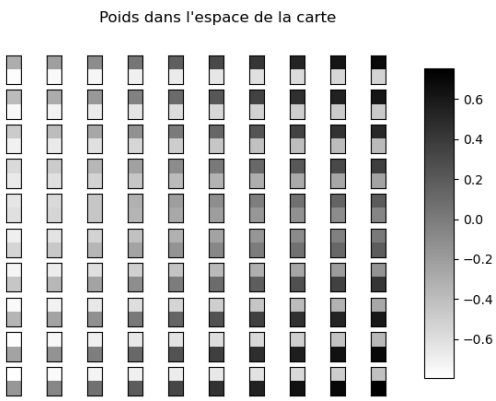
A présent, nous allons modifier les jeu de données pour étudier la répartition des poids des neurones

- Ensemble de données 2

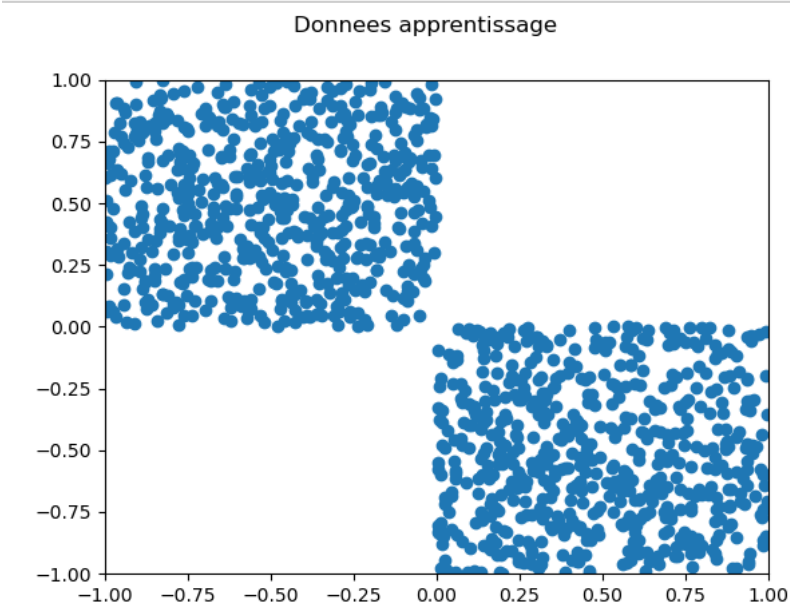


η	σ	N	Taille de la carte	Forme de la carte	Dataset	Erreur de quantification vectorielle	M n o
0.05	1.4	30000	10x10	Carré	Ensemble de données 2	0.035265872682625155	4

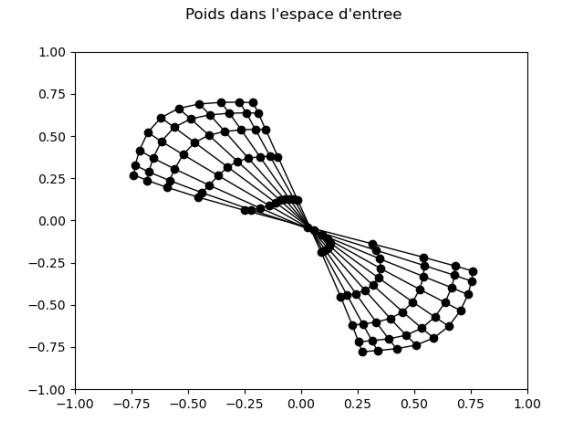


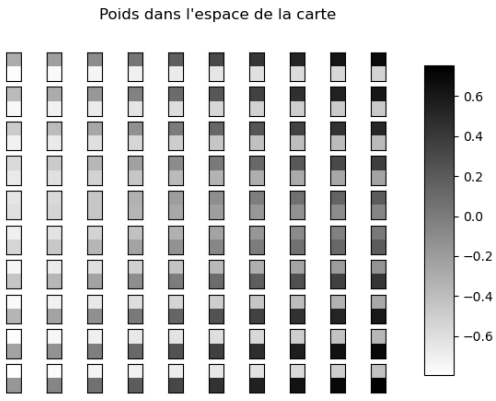


- Ensemble de données 3

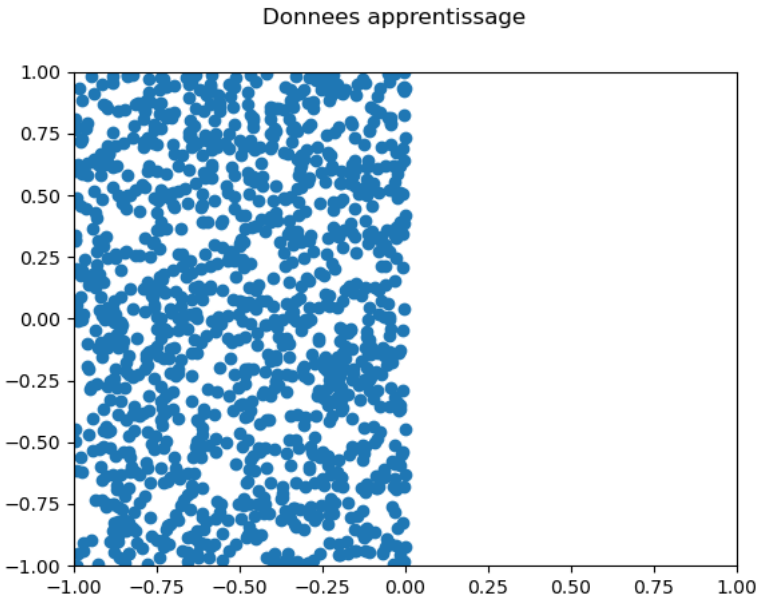


η	σ	N	Taille de la carte	Forme de la carte	Dataset	Erreur de quantification vectorielle	M n o
0.05	1.4	30000	10x10	Carré	Ensemble de données 3	0.037165290760562096	4

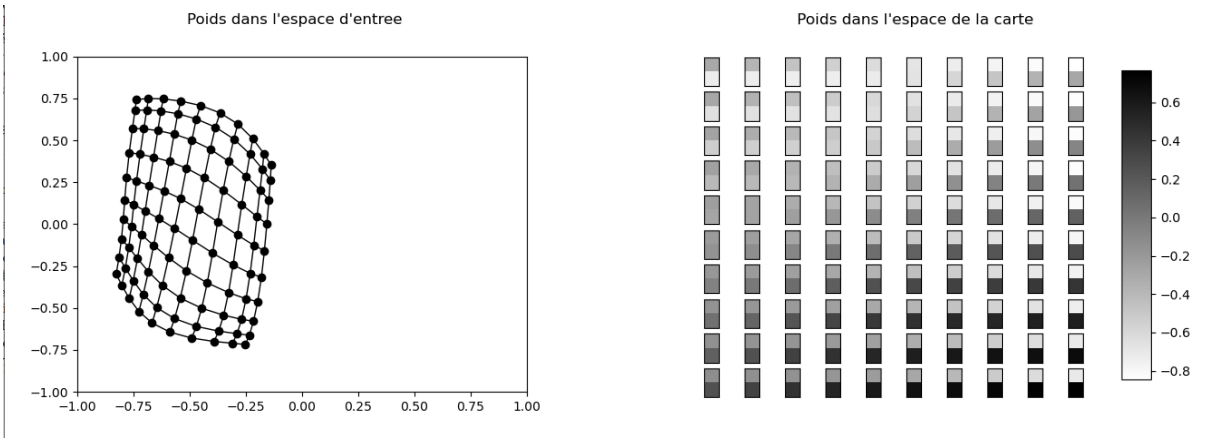




- Ensemble de données personnalisé



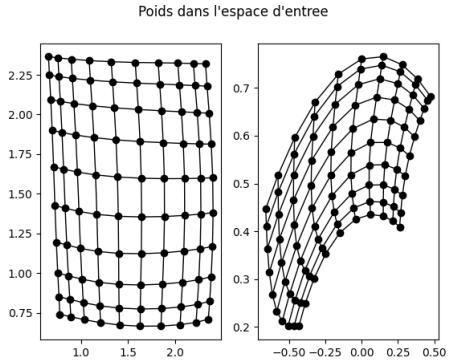
η	σ	N	Taille de la carte	Forme de la carte	Dataset	Erreur de quantification vectorielle	Moyer mesur organi
0.05	1.4	30000	10x10	Carré	Ensemble de données personnalisé	0.03562096890760	177.1



Remarque:
On peut facilement voir que le jeu de données influe sur l'organisation des neurones en fonction de la répartition des points d'entrée. On retrouve les neurones aux endroits où se trouvait les points d'entrée.

4.4 Bras robotique

- 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/justifier le principe.



Le graphique de gauche représente la position motrice du bras $\Rightarrow (\theta_1, \theta_2)$ pour chaque point
Le graphique de droite représente la position du bras dans l'espace $\Rightarrow (x_1, x_2)$ pour chaque point

👉 Position du bras à partir de la position motrice graphique
(graphique gauche → graphique droit)

La position spatiale du bras peut être déterminée à partir d'une position motrice (θ_1, θ_2) en cherchant le neurone le plus proche de ces coordonnées passées en entrée. On va faire la **distance euclidienne** entre ce point en entrée et **tous** les neurones, et choisir la distance la plus faible.

Le neurone choisi aura des coordonnées qui correspondent à (x_1, x_2) dans la position spatiale du bras.

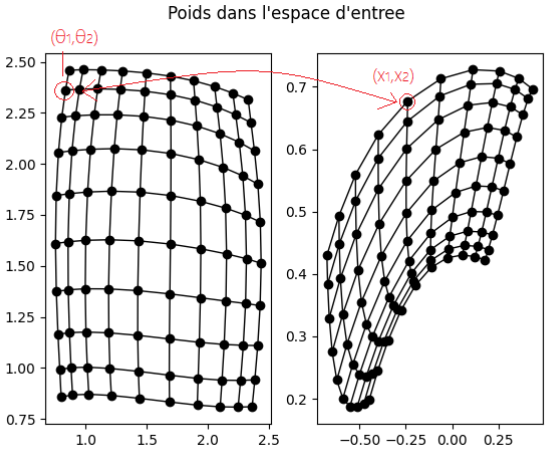
👉 Position motrice à partir de la position du bras à partir
(graphique droit → graphique gauche)

On fait l'inverse :

La position motrice du bras peut être déterminée à partir d'une position spatiale (x_1, x_2) en cherchant le neurone le plus proche de ces coordonnées passées en entrée. On va faire la **distance euclidienne** entre ce point en entrée et **tous** les neurones, et choisir la distance la plus faible.

Le neurone choisi aura des coordonnées qui correspondent à (θ_1, θ_2) dans la position motrice.

Exemple illustré :



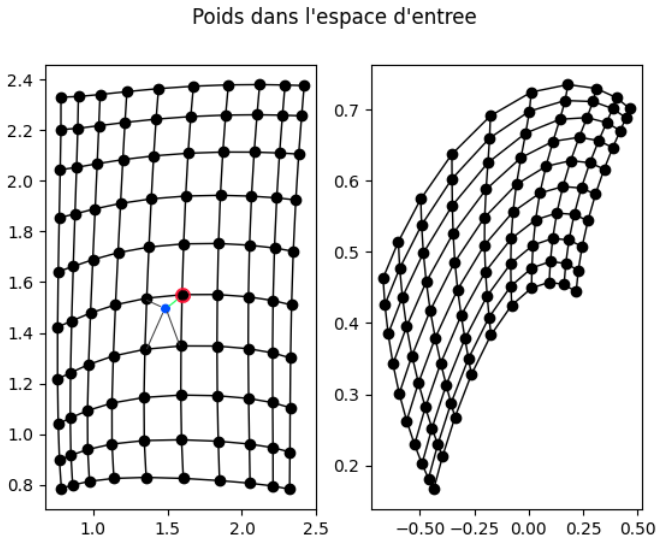
Chaque neurone dans l'espace de position motrice a son équivalent dans l'espace de position spatiale du bras.

Rappel : distance euclidienne : $d(p,q) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2}$ (à faire pour chaque neurone). Nous avons utilisé en Python `numpy.linalg.norm(p - q)`

Notre implémentation : les méthodes `arm_pos_prediction` et `motor_pos_prediction` permettent d'évaluer la position du neurone le plus proche d'un point donné.

Exemple d'exécution avec le point $(\theta_1 = 1.5, \theta_2 = 1.5)$, soit le milieu de la carte environ :

```
Position spatiale du bras estimée pour le point (θ1,θ2) = (1.5,1.5) => (x1, x2) = (1.597722219593734,1.5507147939474044)
Position motrice estimée pour le point (x1,x2) = (1.5,1.5) => (θ1, θ2) = (0.8479218957967414,1.2424144854745778)
```



Point bleu : notre point d'entrée $(\theta_1 = 1.5, \theta_2 = 1.5)$

Ligne verte : distance la plus courte parmi tous les neurones

Point rouge : neurone le plus proche sélectionné

Remarque : normalement le point bleu doit être relié à tous les neurones pour trouver la distance la plus courte.

- De quel autre modèle vu en cours se rapproche cette méthode (apprentissage sur le quadruplet pour ensuite en retrouver une sous partie étant donnée l'autre) ? Quels auraient été les avantages et les inconvénients d'utiliser cette autre méthode ?

👉 Cette méthode peut être approchée par la méthode du gaz neuronal - Martinetz et Schulten qui présente beaucoup de similarités.
+ Avantage : Contrairement à Kohonen, la forme de carte n'est pas une contrainte pour la quantification vectorielle des données.
− Inconvénient : plus sensible au bruit par dessus les données passées en entrée.

- Si l'objectif était de prédire uniquement la position spatiale à partir de celle motrice, quel autre modèle du cours aurait-on pu utiliser ? Quels auraient été les avantages et les inconvénients ?

👉 On aurait pu utiliser la méthode du Perceptron multi-couches - Werbos & Rumelhard.

Par couche, chaque neurone reçoit les sorties de tous les neurones de la couche précédente.

+ Avantage : plus simple à implémenter, les données en sortie requièrent moins de traitement

− Inconvénient : le modèle a des chances de diverger, nous ne somme pas sûrs de converger vers le résultat attendu

- On veut déplacer le bras d'une position motrice (θ_1, θ_2) à une nouvelle position (θ'_1, θ'_2). En utilisant au maximum 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/justifier le principe et implémentez le.

👉 On peut utilise l'algorithme de Dijkstra pour trouver le plus court chemin entre deux neurones (θ_1, θ_2) et (θ'_1, θ'_2). On pourra ainsi prédire les positions spatiales prises par la main.