Intelligence Artificielle

Réseaux de neurones

Année 2021 – 2022

Réalisé par :

FERCHIOU Iskander
OULED MOUSSA Yanis

Responsable de l'UE :

M. LEFORT Mathieu

POLYTECH Lyon - UNIVERSITE CLAUDE BERNARD LYON 1
Domaine Scientifique de La Doua – 15, Boulevard Latarjet
69622 VILLEURBANNE CEDEX
Tél. (33) 04.72.43.12.24 - Fax. (33) 04.72.43.12.25
http://polytech.univ-lyon1.fr/

Sommaire

Etude théorique

Influence de η

On notera l'entrée courante X et le vecteur de poids courant du neurone gagnant W^*.

- *Dans le cas où $\eta = 0$ quelle sera la prochaine valeur des poids du neurone gagnant ?*

Si η = 0, la prochaine valeur des poids du neurone gagnant sera inchangée car la variation des poids ΔW_{j^*} est nulle.

$\Delta W_{j^*} = 0$

$\Leftrightarrow W_{j^*}' - W_{j^*} = 0$ où W_{j^*}' correspond à la prochaine valeur des poids du neurone gagnant.

$\Leftrightarrow W_{j^*}' = W_{j^*}$

- *Même question dans le cas où $\eta = 1$*

Dans le cas où $\eta = 1$, l'équation sera de la forme :

$$\Delta W_{j^*} = V(j^*, j^*)\left(X - W_{j^*}\right)$$

$$= e^{\frac{-||j^* - j^*||_c^2}{2\sigma^2}}\left(X - W_{j^*}\right)$$

$$= X - W_{j^*}$$

Or, $\Delta W_{j^*} = W_{j^*}' - W_{j^*}$ où W_{j^*}' correspond à la prochaine valeur des poids du neurone gagnant. Par conséquent, on a :

$\Delta W_{j^*} = X - W_{j^*}$

$\Leftrightarrow W_{j^*}' - W_{j^*} = X - W_{j^*}$

$\Leftrightarrow W_{j^*}' = X$

Le poids du neurone gagnant W_{j^*}' aura donc la valeur de l'entrée X.

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

$\Delta W_{j^*} = W_{j^*}' - W_{j^*}$

$\Leftrightarrow W_{j^*}' = \Delta W_{j^*} + W_{j^*}$

$$= \eta e^{\frac{-||j^* - j^*||_c^2}{2\sigma^2}}\left(X - W_{j^*}\right) + W_{j^*}$$

$$= \eta\left(X - W_{j^*}\right) + W_{j^*}$$

$$= \eta X - \eta W_{j^*} + W_{j^*}$$

$$= \eta X - \eta W_{j^*} + W_{j^*}$$

$$= \eta X + W_{j^*}(1 - \eta)$$

Plus η est proche de 1, plus la valeur se rapprochera de X. Si η tend vers 0, alors la valeur ne sera pas très loin de W_{j^*}.

- *Que va-t-il se passer si $\eta > 1$?*

D'après la question précédente, si $\eta > 1$, on aura : $\eta X > X$ et $W_{j^*}(1-\eta) \leq 0$. La valeur du nouveau poids dépassera l'entrée courante X.

Influence de σ

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

$\sigma_2 > \sigma_1$

$\Leftrightarrow -2{\sigma_2}^2 < -2{\sigma_1}^2$

$\Leftrightarrow \frac{-\left||j-j^*|\right|_c^2}{2{\sigma_2}^2} > \frac{-\left||j-j^*|\right|_c^2}{2{\sigma_1}^2}$

$\Leftrightarrow e^{\frac{-\left||j-j^*|\right|_c^2}{2{\sigma_2}^2}} > e^{\frac{-\left||j-j^*|\right|_c^2}{2{\sigma_1}^2}}$

$\Leftrightarrow \eta e^{\frac{-\left||j-j^*|\right|_c^2}{2{\sigma_2}^2}}(X - W_j) > \eta e^{\frac{-\left||j-j^*|\right|_c^2}{2{\sigma_1}^2}}(X - W_j)$

$\Leftrightarrow \Delta W_{j_{\sigma_2}} > \Delta W_{j_{\sigma_1}}$

Ainsi, on constate que la variation des poids augmente si σ augmente. De ce fait, les neurones proches du neurone gagnant vont plus apprendre de l'entrée courante.

- *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" ?*

En s'appuyant sur la question précédente, si σ est plus grand alors l'auto-organisation obtenue sera plus « resserrée ». En effet, l'augmentation de la variation des poids va engendrer un rapprochement des neurones en direction du neurone gagnant.

- *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 ?*

Afin de mesurer l'influence de σ sur le comportement de l'algorithme, on pourrait quantifier ce phénomène en estimant la dispersion des valeurs, c'est-à-dire, en calculant la variance des distances entre les poids des neurones. Plus cette valeur est grande, plus l'auto-organisation obtenue est lâche. Inversement, plus l'écart par rapport à la moyenne est petit, plus l'auto-organisation obtenue est resserrée. Voici une formulation mathématique de cette proposition :

$$V = \frac{1}{\frac{n(n-1)}{2}} \sum_{i=1}^{n_A} \sum_{j=i+1}^{n_b} (d(i,j) - \overline{d})^2 = \frac{2}{n(n-1)} \sum_{i=1}^{n_A} \sum_{j=i+1}^{n_b} (d(i,j) - \overline{d})^2$$

Où :

- n correspond au nombre total de neurones ($n_A \times n_b$).
- $d(i,j)$ est la distance entre deux neurones i et j.
- $\frac{n(n-1)}{2}$ est le nombre de distances.
- $\overline{d}$ représente la moyenne des distances.

Influence de la distribution d'entrée

- *Prenons le cas très simple d'une carte à 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 fiable - 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) ?*

Si les deux entrées lui sont présentées autant de fois, alors le vecteur du poids du neurone devrait converger vers le point le plus au centre du segment rattachant les deux entrées.

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

Dans ce cas, le neurone devrait être n fois plus proche de X_1 que de X_2 sur le segment reliant les 2 entrées.

- *En déduire comment vont se répartir les neurones en fonction de la densité des données dans le cas (normal) d'une carte à plusieurs neurones recevant des données d'une base d'apprentissage. Pour rappel, la mesure de quantification vectorielle permet de mesurer en partie ce phénomène.*

Les neurones devraient se répartir dans l'espace pour coller au mieux à la fonction de densité des données d'entrée et donc en prendre sa forme. Pour que les neurones puissent en prendre correctement la forme, il faut que la carte soit compatible avec la forme des données d'entrée.

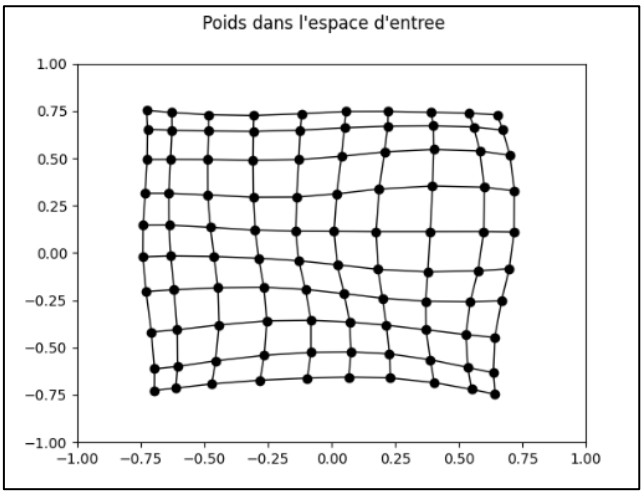
Etude pratique

Implémentation

```
44      def compute(self,x):
45          self.y = numpy.linalg.norm(self.weights - x)
```

```
53      def learn(self,eta,sigma,posxbmu,posybmu,x):
54          self.weights[:] = self.weights + eta * numpy.exp(
55                  -((self.posx - posxbmu) ** 2 + (self.posy - posybmu) ** 2) / (2 * sigma ** 2)
56              ) * (x - self.weights)
```

En effectuant l'apprentissage sur le jeu de données 1 avec les paramètres de base, on obtient bien :

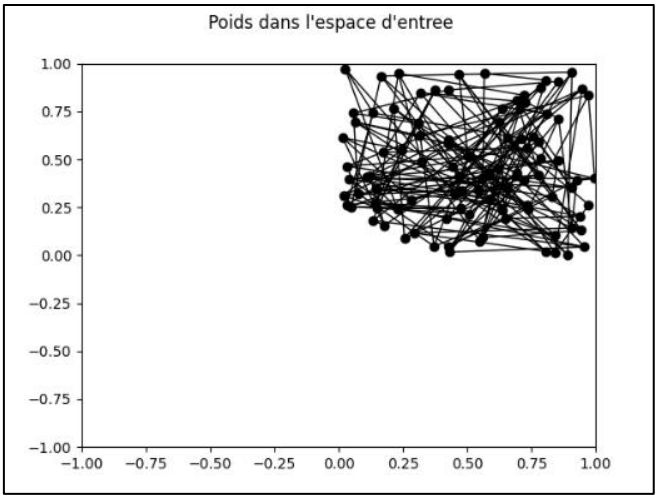


Analyse de l'algorithme

- *Etudiez l'influence des éléments suivants sur le fonctionnement de l'algorithme de Kohonen (qualitativement et quantitativement avec la mesure d'erreur de quantification vectorielle fourni et avec la mesure d'auto-organisation que vous avez proposé à la section 3.2)*

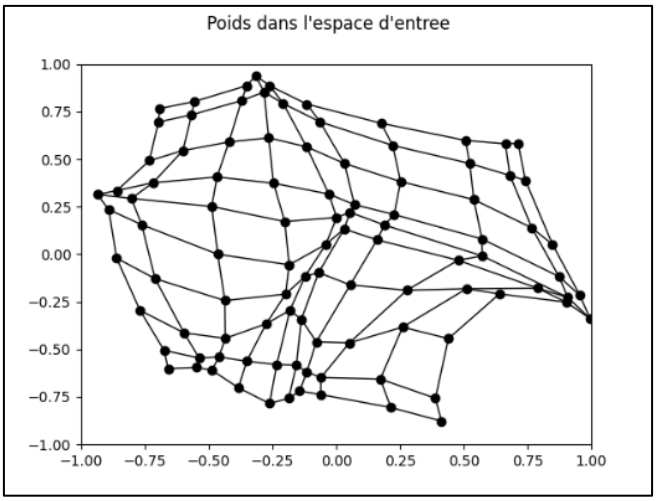
    - *Taux d'apprentissage η*

Si η = 0, l'apprentissage ne s'effectue pas et les neurones ne bougent pas, ce qui aboutit à une auto-organisation plutôt resserré par défaut.



```
erreur de quantification vectorielle moyenne  0.4026007249507723
mesure d'auto-organisation du réseau  0.060598768684924725
```

Si η = 1, l'apprentissage est systématique mais le réseau rencontre des difficultés au niveau de la stabilisation.

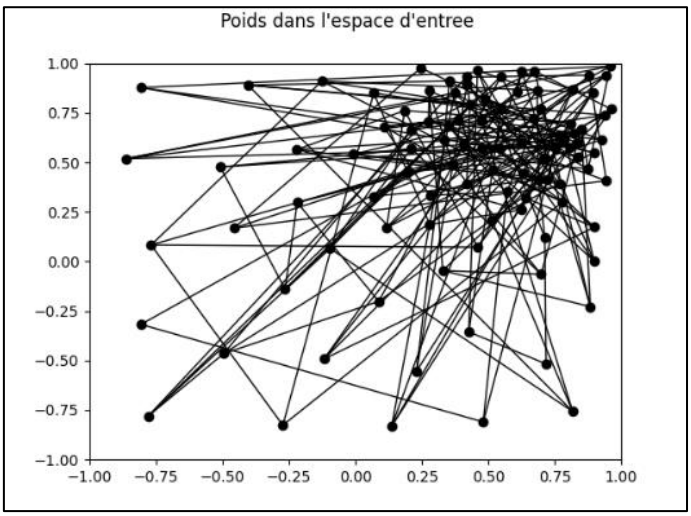



```
erreur de quantification vectorielle moyenne  0.03071305254881645
mesure d'auto-organisation du réseau  0.17965051174465246
```

De plus, on remarque que : plus la valeur de η augmente, plus l'erreur de quantification vectorielle moyenne diminue. Dans ce cas précis, la variance des distances entre les poids des neurones augmente également, et par conséquent, le type d'auto-organisation est plus lâche. Toutefois, les neurones sont beaucoup mieux répartis sur la grille avec un η proche de 1 par rapport à un η qui tend vers 0.

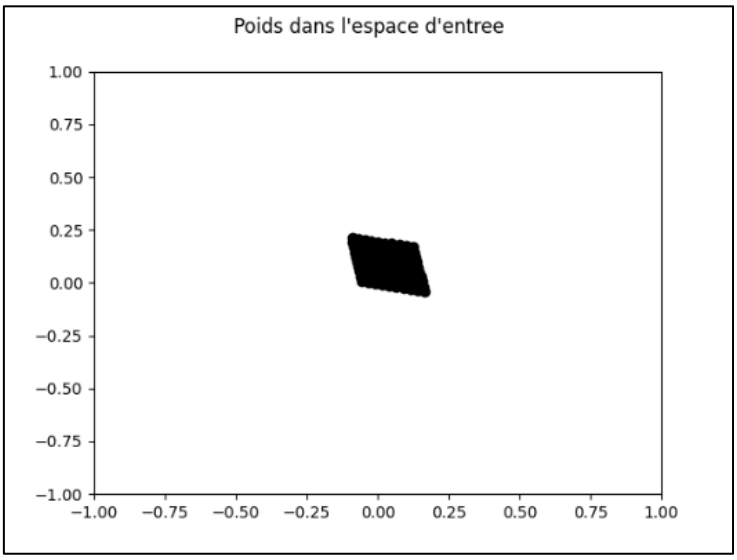
- *Largeur du voisinage σ*

Pour σ = 0.1, les neurones sont répartis de façon anarchique car leurs déplacements sont limités. On observe une erreur de quantification vectorielle moyenne assez faible et une auto-organisation très lâche.




```
erreur de quantification vectorielle moyenne  0.01381500791506828
mesure d'auto-organisation du réseau  0.2359203678773223
```

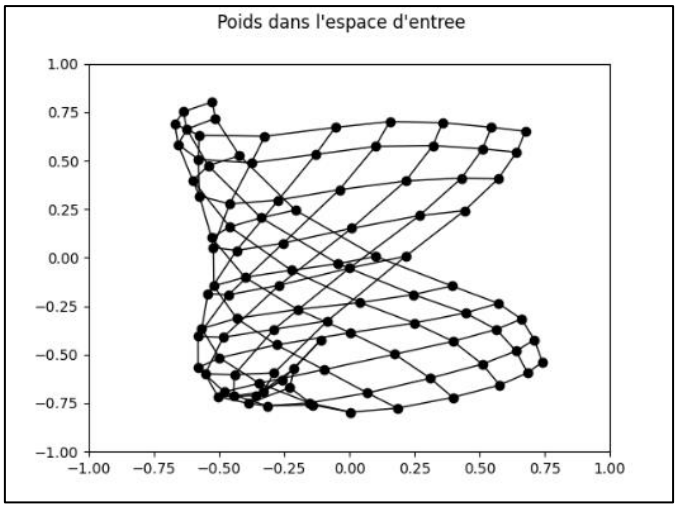
Pour σ = 10, l'auto-organisation est très resserée car les neurones proches du neurone gagnant apprenent beaucoup plus de l'entrée courante. Contrairement au taux d'apprentissage, une augmentation de la largeur du voisinage provoque une hausse de l'erreur de quantification vectorielle.




```
erreur de quantification vectorielle moyenne  0.4889947478926506
mesure d'auto-organisation du réseau  0.0031391315611111772
```

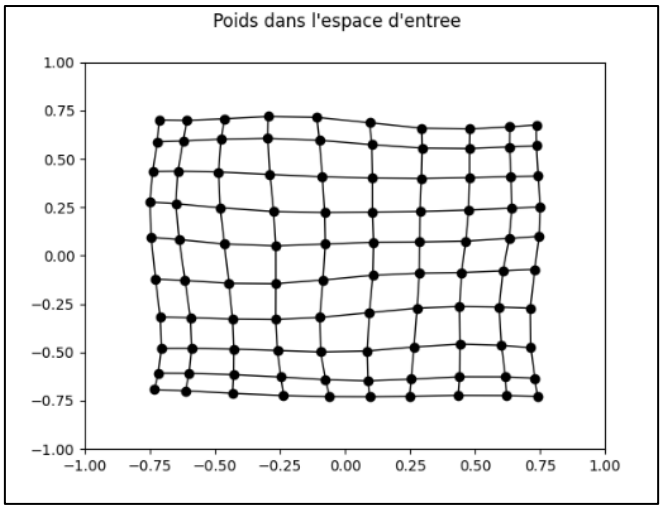
- *Nombre de pas de temps d'apprentissage N*

En prenant une valeur de N inférieure à celle par défaut (par exemple N = 3 000), on constate que la carte n'a pas eu le temps de totalement se mettre en place.




```
erreur de quantification vectorielle moyenne  0.029256642764744 72
mesure d'auto-organisation du réseau  0.16799205459045563
```

En prenant une valeur de N supérieur à celle par défaut (par exemple N = 50 000), on voit que les résultats sont très similaires à ceux de base avec N = 30 000 et que les changements sont négligeables donc il n'est pas nécessaire d'augmenter autant le nombre de pas de temps d'apprentissage.



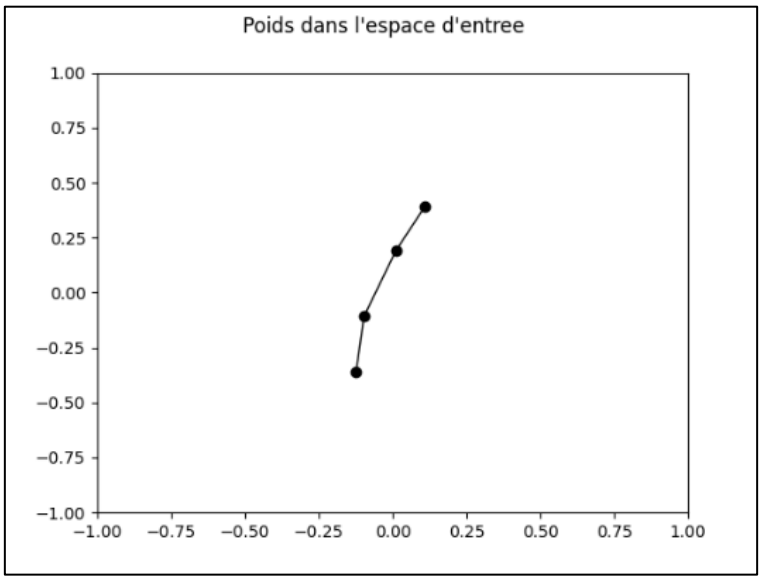

```
erreur de quantification vectorielle moyenne  0.021298667960850157
mesure d'auto-organisation du réseau  0.1555776426472397
```

- *Taille et forme de la carte (vous pouvez tester facilement des formes 'lignes', 'carrées' et 'rectangles')*

Ici, il suffit d'agir sur le paramètre « $gridsize$ » du constructeur du réseau afin de modifier la taille et la forme de la carte. Pour bien comprendre l'influence de cette variable sur le fonctionnement de l'algorithme de Kohonen, nous avons tester des petites, grandes et « très grandes » tailles pour chacune des formes.

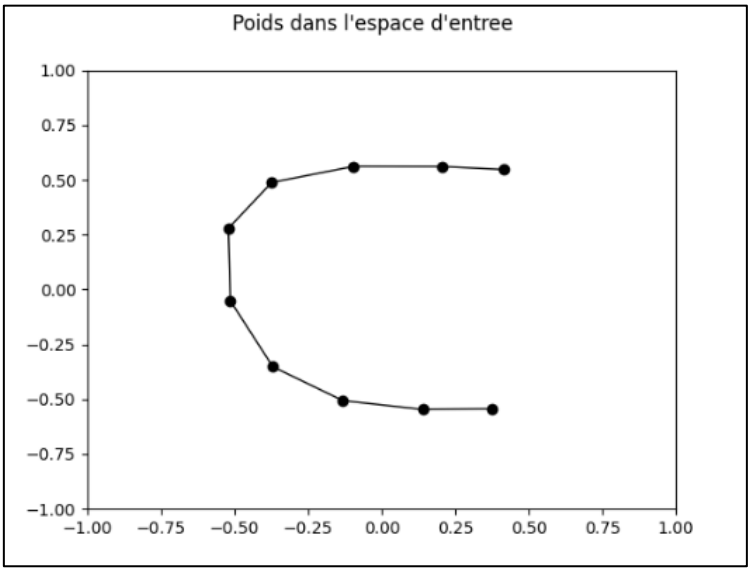
➢ Formes lignes :

Petite ligne (4x1)



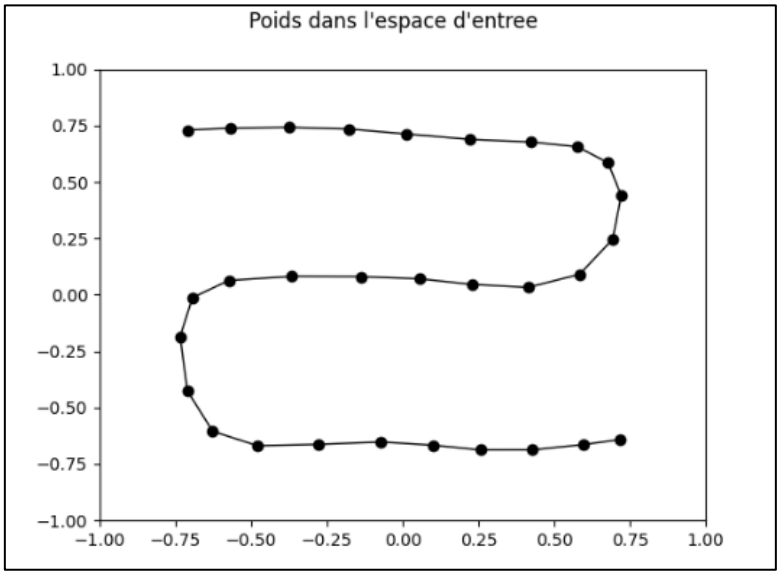

```
erreur de quantification vectorielle moyenne  0.4324150593505731
mesure d'auto-organisation du réseau  0.03560403288146996
```

Grande ligne (10x1)




```
erreur de quantification vectorielle moyenne  0.1503201503237185
mesure d'auto-organisation du réseau  0.1005308336655413
```

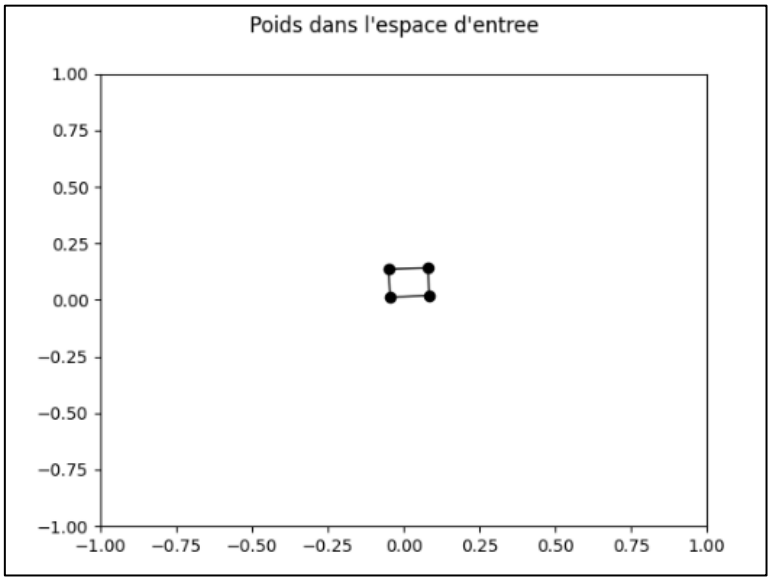
Très grande ligne (30x1)




```
erreur de quantification vectorielle moyenne  0.046969661829031555
mesure d'auto-organisation du réseau  0.1828186214713808
```

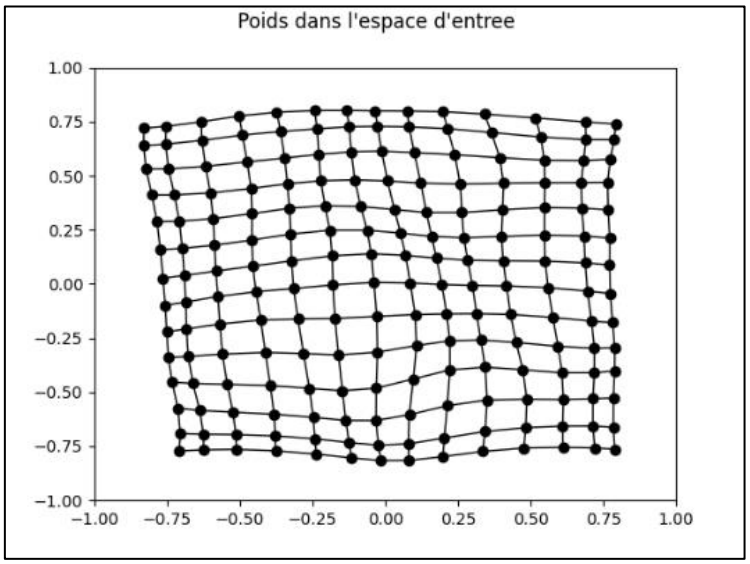
- Formes carrées :

Petit carrée (2x2)



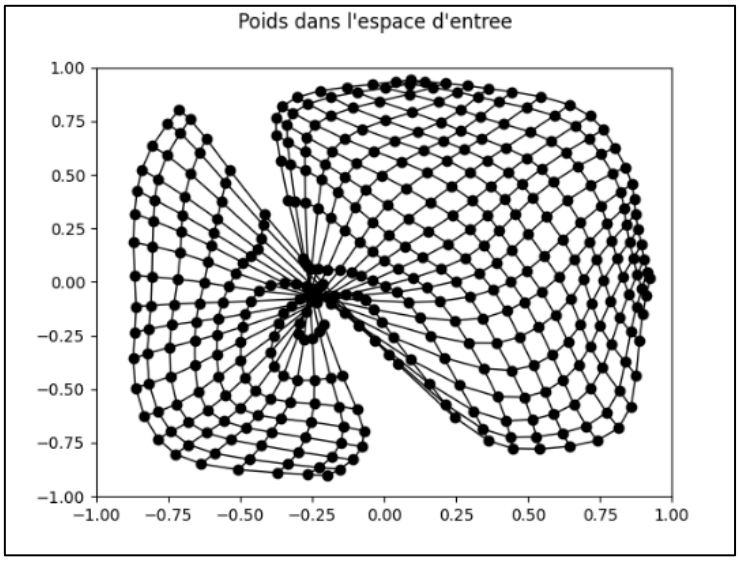

```
erreur de quantification vectorielle moyenne  0.5746035821675614
mesure d'auto-organisation du réseau  0.0011113613191323023
```

Grand carrée (14x14)




```
erreur de quantification vectorielle moyenne  0.010982578483766388
mesure d'auto-organisation du réseau  0.17932370645041992
```

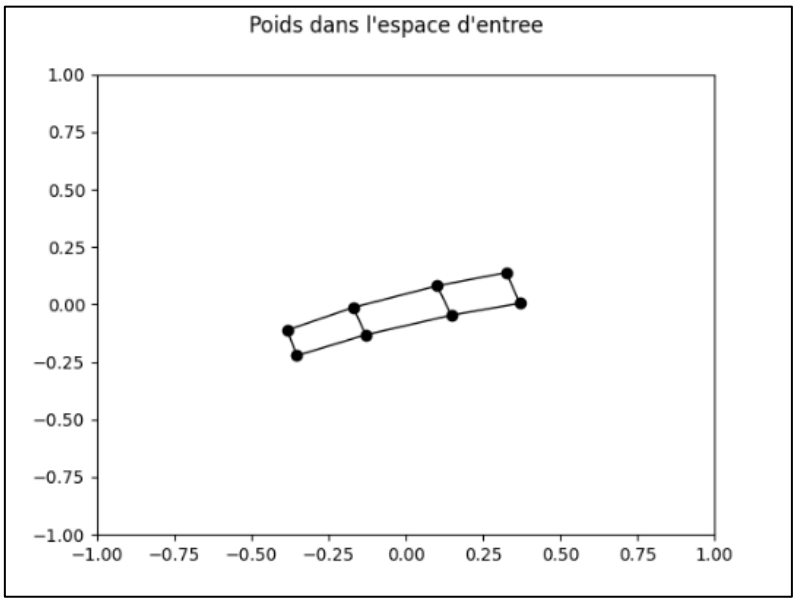
Très grand carrée (20x20)




```
erreur de quantification vectorielle moyenne  0.0046818887289389625
mesure d'auto-organisation du réseau  0.1876416901393755
```

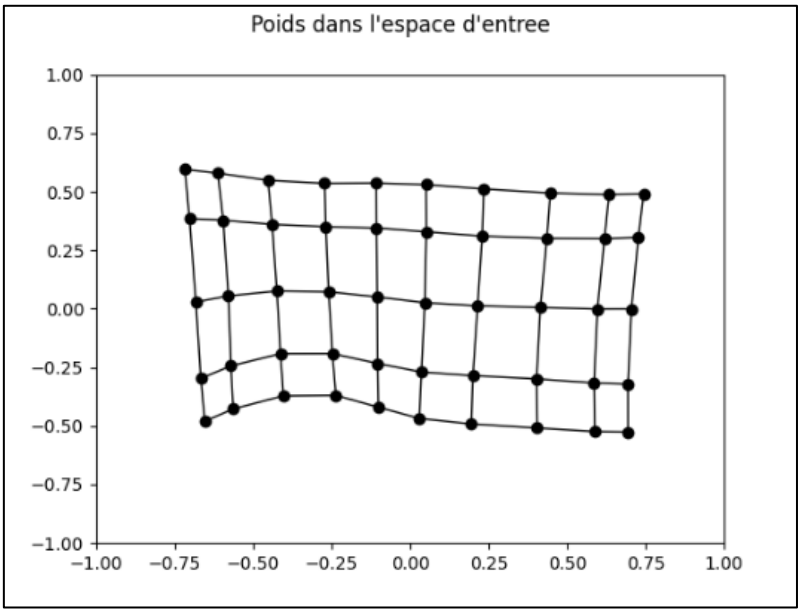
- Formes rectangles :

Petit rectangle (4x2)



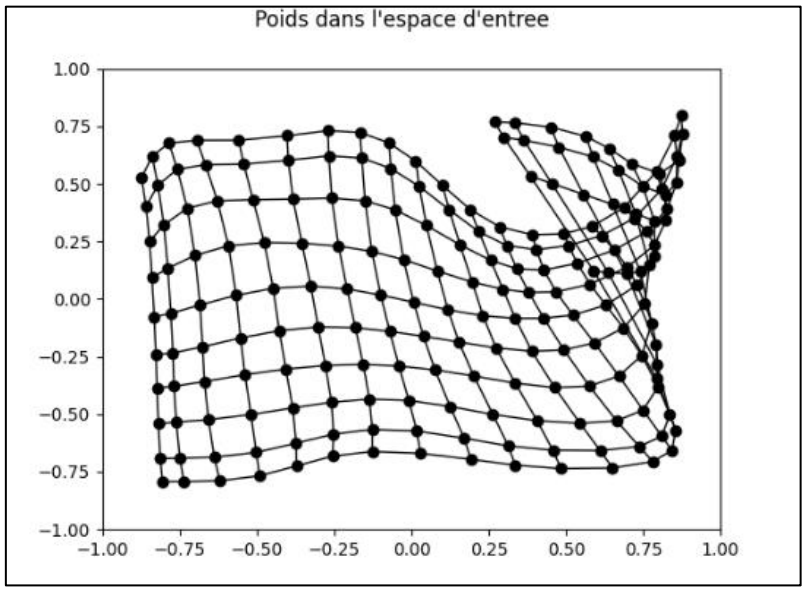

```
erreur de quantification vectorielle moyenne  0.3720411602139891
mesure d'auto-organisation du réseau  0.05404234189519639
```

Grand rectangle (10x5)




```
erreur de quantification vectorielle moyenne  0.055276219560319986
mesure d'auto-organisation du réseau  0.1276270679101134
```

Très grand rectangle (20x10)




```
erreur de quantification vectorielle moyenne  0.011760669061010426
mesure d'auto-organisation du réseau  0.19966512809749526
```

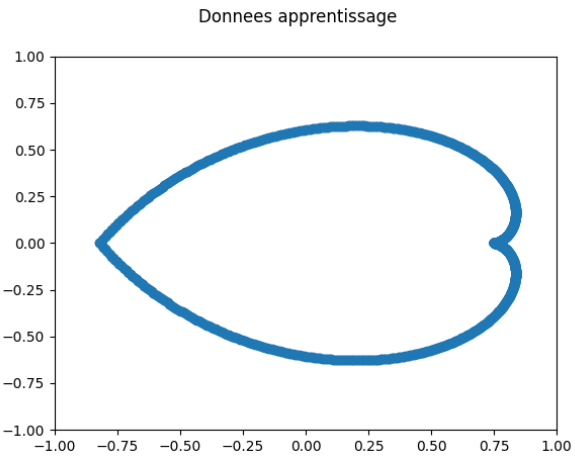
Peu importe la forme choisie, on remarque que la modification de la taille affecte nos résultats :

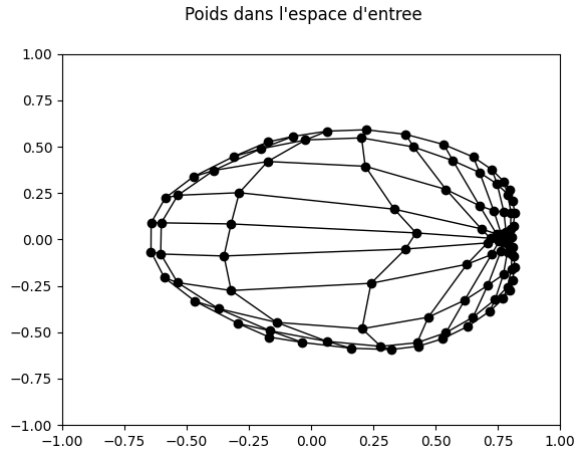
- Plus la taille est grande, plus la forme choisie est difficilement « distinguable » visuellement. En effet, les neurones cherchent à maximiser l'espace occupé en se déformant.
- Plus la taille est grande, plus l'erreur de quantification vectorielle moyenne est faible.
- Plus la taille est grande, plus la mesure d'auto-organisation du réseau est grande. Par conséquent, l'auto-organisation est de plus en plus lâche.
- Plus la taille est grande, plus l'algorithme de Kohonen met du temps à se terminer.

Remarque : la réciproque de ces observations est vraie.

- *Jeux de données. En particulier créez vos propres jeux de données avec des données non uniformément distribuées pour étudier la répartition des poids des neurones.*

Jeu de données en cardioïde

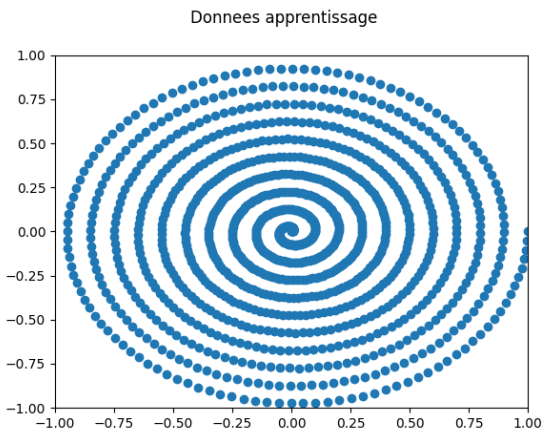


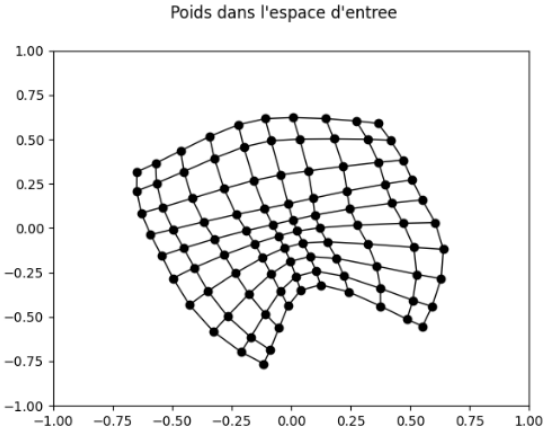



```
erreur de quantification vectorielle moyenne  0.0025889429224656957
mesure d'auto-organisation du réseau  0.16526556005585066
```

Dans le cas d'une cardioïde, on observe que le réseau est capable de reproduire correctement sa forme d'origine. L'erreur de quantification vectorielle moyenne est assez faible et la mesure d'auto-organisation du réseau atteste d'une disposition plus « lâche » que « resserré » des poids des neurones.

Jeu de données en Spirale





```
erreur de quantification vectorielle moyenne  0.020928638486935224
mesure d'auto-organisation du réseau  0.08093571551029488
```

Dans le cas d'une spirale, on obtient après exécution une forme similaire à celle généré par le premier jeu de données et les paramètres par défaut. La précision du résultat est satisfaisante au vu de l'erreur de quantification vectorielle moyenne et l'auto-organisation est plus « resserrée » que « lâche » d'après la mesure d'auto-organisation du réseau.

Bras robotique

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

Théorie

Pour prédire une position (x_1, x_2) du bras étant donné une position motrice (θ_1, θ_2), il est possible d'utiliser les formules suivantes :

$$x_1 = l_1 * \cos(\theta_1) + l_2 * \cos(\theta_1 + \theta_2), x_2 = l_1 * \sin(\theta_1) + l_2 * \sin(\theta_1 + \theta_2)$$

Cependant, cette méthode ne permettrait pas de retrouver la position motrice (θ_1, θ_2) à partir de la position spatiale (x_1, x_2). De plus, il s'agit d'une déduction mathématique alors que nous recherchons une solution mettant en œuvre la carte de Kohonen.

Par conséquent, il faudrait trouver une solution usant de la position des neurones dans la carte. Pour cela, on pourrait calculer les distances qui séparent tous les neurones avec l'entrée et de prendre le neurone le plus proche. Cette méthode permettrait donc de prédire un sous-ensemble du quadruplet $(\theta_1, \theta_2, x_1, x_2)$ à partir de l'autre sous-ensemble.

Implémentation

Ainsi, nous avons implémenté cette méthode et en voici les résultats :

```
Position du bras : (x1, x2), [1 1] - Position motrice estimée : (t1, t2), [1.13470352 0.93484536]
Position motrice : (t1, t2), [0.75 0.25] - Position du bras estimée : (x1, x2), [0.77185257 0.50008608]
```

Comme vous le voyez ci-dessus, nous retrouvons bien une estimation des positions manquantes pour un sous-ensemble donné du quadruplet initial.

- *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 se rapproche du modèle du réseau de Hopfield. En effet, il est possible de « dégrader » le réseau de neurones formés par le quadruplet $(\theta_1, \theta_2, x_1, x_2)$ en retirant une sous-partie (position spatiale ou commande motrice) puis en reconstruisant cette dernière grâce aux données mémorisées. En procédant ainsi, on pourrait obtenir des résultats très satisfaisants mais la limite de stockage du nombre d'entrées pourrait être contraignante.

- *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 ?*

Dans ce cas précis, on aurait pu faire appel à la méthode du Perceptron multicouches en passant en entrée la position motrice pour récupérer la position spatiale en sortie.

Ce modèle possède l'avantage d'approximer globalement des fonctions de manière incrémentale et peut également produire de très bons résultats lorsqu'il est paramétré correctement. Il est aussi possible de l'utiliser dans le cas d'étude de grandes quantités de données.

Cependant, ce modèle n'est tout de même pas parfait. En effet, il n'est pas évident à mettre en place, car son paramétrage peut devenir assez complexe puisque l'on ne peut pas prévoir son comportement dans les couches de neurones cachées. De plus, un grand nombre de neurones peut être généré en

raison du risque de sur-apprentissage. Enfin, on manque de garantie en matière de convergence et d'efficacité de ce modèle.

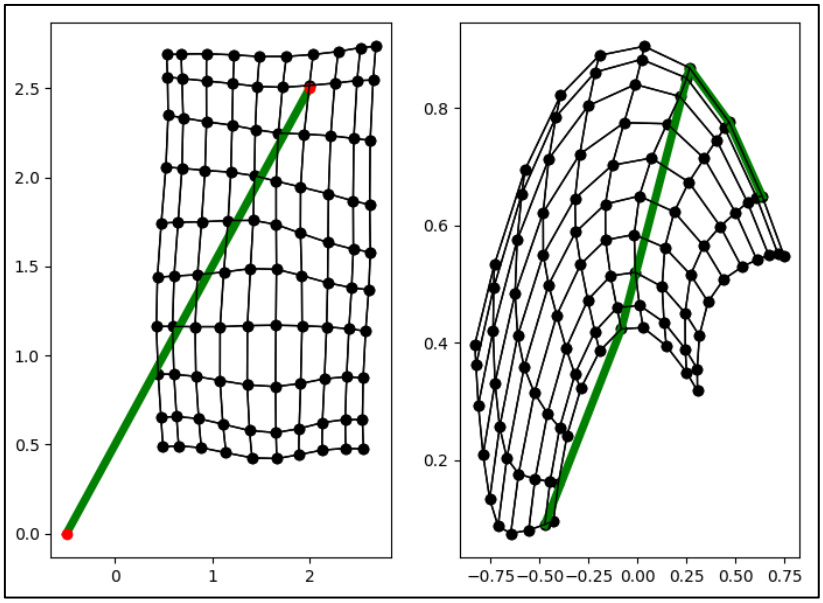
- *On veut déplacer le bras d'une position motrice (θ_1, θ_2) à une nouvelle (θ'_1, θ'_2). En utilisant au maximum la carte apprise, comment prédire la suite des positions spatiales prise par la main ? On demande ici de pouvoir tracer grossièrement la trajectoire, pas forcément d'avoir la fonction exacte de toutes les positions prises. Expliquer/justifier le principe et implémentez le.*

Théorie

Pour prédire la suite des positions spatiales prise par la main, il est possible de tracer une droite avec les deux positions motrices (θ_1, θ_2) et (θ'_1, θ'_2) puis de calculer les positions spatiales des points se trouvant sur la droite en utilisant la méthode implémentée précédemment.

Implémentation

Ainsi, nous avons implémenté cette proposition et voici le résultat renvoyé par le programme :



Ici, nous avons choisi les positions motrices (-0.5, 0.0) et (2.0, 2.5) pour pouvoir créer la droite sur le graphique de gauche. Ensuite, comme expliqué précédemment, nous avons simplement recherché la position spatiale de chaque point situé sur la droite tracée, ce qui donne l'affichage observée à droite.