BRUNEAU Richard
Technique d'IA
Juin 2020

Document réponses

Etude « théorique » de cas simples

Influence de η

Si η est égale à 0, alors $\Delta W_{i,j} = 0$. Du coup, la prochaine valeur du neurone gagnant sera la valeur du neurone gagnant initialement. Cette valeur ne varie pas.

Si η est égale à 1, alors $\Delta W_{i,j} = 1*1*(x_i - w_{i,j})$. Du coup, la prochaine valeur du poids va tendre vers X.

Si η est compris entre]0,1[, alors $W_{i,j} = \eta*1*(x_i - w_{i,j})$. Du coup, si η tend vers 0, la valeur du neurone restera proche de W* et si η tend vers 1, la valeur va tendre vers X.

Influence de σ

Si σ augmente, alors la division va tendre vers 0. Nous allons donc avoir la fonction de voisinage qui tend vers 1. Cela a pour effet que les voisins vont plus apprendre l'entrée courante.

Plus σ est grand, plus l'auto-organisation obtenue sera resserrée. En effet, plus le sigma augmente, plus les neurone apprennent et donc plus ils se déplacent.

Une mesure envisageable pour quantifier l'influence de σ serait de mesurer l'aire de la grille en fonction de différents σ et d'essayer de trouver un rapport entre la différence de l'air et la variation du sigma. Si nous prenons 4 points avec pour coordonnées :

A = (Xmin, Ymax), le point en haut à gauche
B = (Xmax, Ymax), le point en haut à droite
C = (Xmin, Ymin), le point en bas à gauche
et D = (Xmax, Ymin), le point en bas à droite.
La formule de l'aire est alpha*beta avec :
alpha = sqrt((XB - XA)^2+(YB - YA)^2) = sqrt((Xmax - Xmin)^2+(Ymax - max)^2)
beta = sqrt((XC - XA)^2+(YC - YA)^2) = sqrt((min - Xmin)^2+(Ymax - Ymin)^2)

Influence de la distribution d'entrée

Le vecteur va converger entre les deux entrées à égale distance de l'une que de l'autre.

Si X1 est présenté n fois plus que X2, alors le neurones va se déplacer vers X1. A 1/n de la mi-distance entre X1 et X2.

Si il est deux fois plus présents, le neurones sera à 1/4 de la distance totale.

Si il est trois fois plus présents, le neurones sera à 1/3 de la mi-distance.

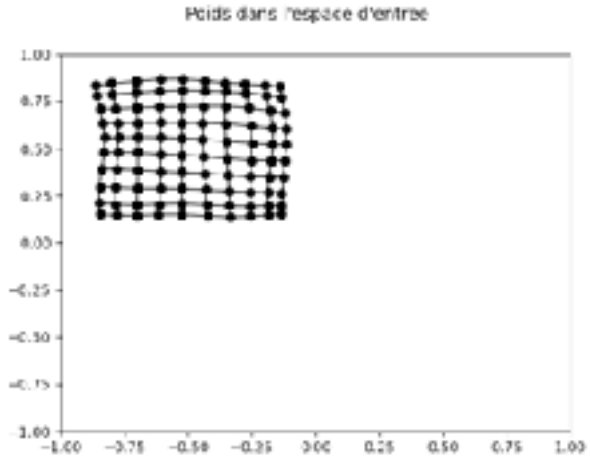
Les neurones vont se placer à l'intérieur de la plage de données de façons à couvrir précisément le centre de la plage.

Étude pratique

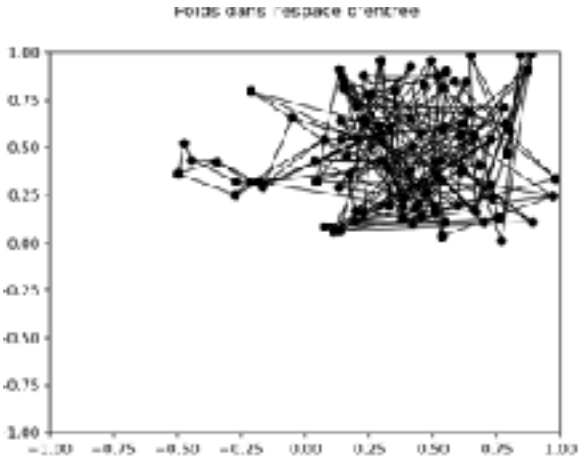
Analyse de l'algorithme

Analyse du taux d'apprentissage de η

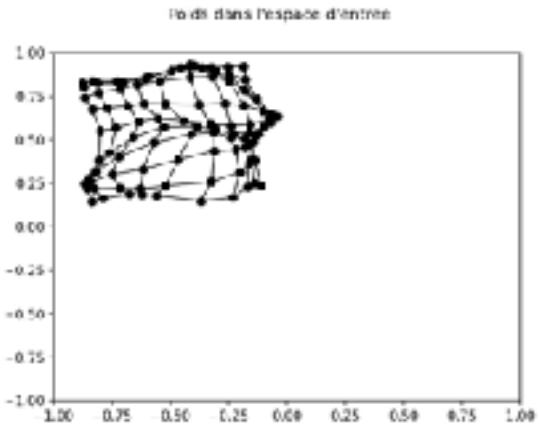
Pour l'analyse du taux d'apprentissage de η, j'ai réalisé ces trois captures d'écran ci-dessous. Sur la première, nous avons η avec la valeur initiale à savoir 0.05. Sur la deuxième capture, η est à une valeur proche de 0, à savoir 0,0001. Enfin sur la dernière, η est à 0,99.



Grille avec les valeurs initiales



Grille avec η à 0,0001



Grille avec η à 0,99

Pour un η à 0,05, l'aire de la grille est de 0,5653 et l'erreur de quantification vectorielle moyenne est de 0,0045.

Pour un η à 0,0001, l'aire de la grille est de 1,4539 et l'erreur de quantification vectorielle moyenne est de 0,8508.

Pour un η à 0,99, l'aire de la grille est de 0,6566 et l'erreur de quantification vectorielle moyenne est de 0,0050.

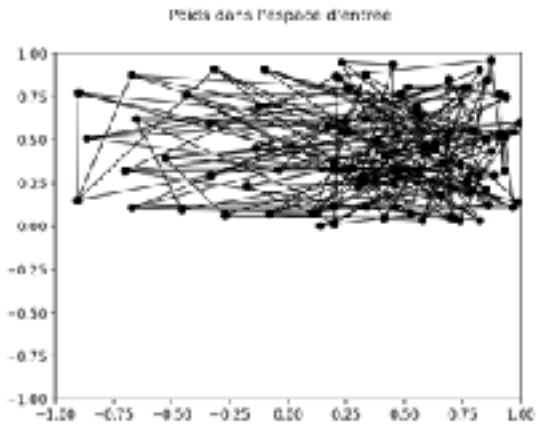
Nous pouvons constater plusieurs choses, la première, c'est que comme prévu avec un η proche de 0, il y a très peu de variations sur la grille de points. La grille est complètement désorganisée comme vous pouvez le constater sur la capture d'écran.

Ensuite, avec un η proche de 1, la grille est plus ou moins bien formée. Ceci est dû au fait qu'il y a beaucoup de mouvements entre chaque pas de temps. On peut constater cela au cours de la simulation.

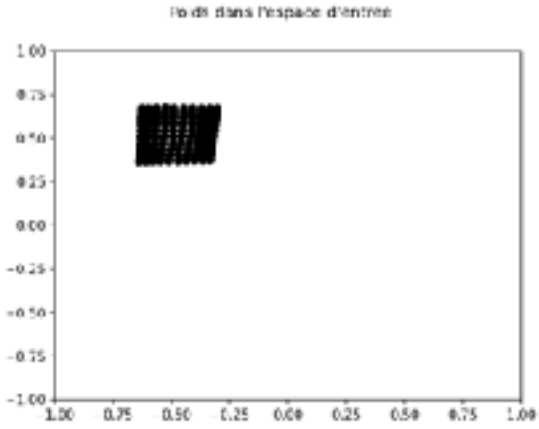
Enfin, l'erreur de quantification vectorielle moyenne augmente en réduisant η.

Analyse de la largeur du voisinage σ

Pour l'analyse de σ, j'ai à nouveau réalisé des captures, les deux nouvelles viennent compléter la première. Celle de gauche est avec un σ à 0,0001 tant dit que celle de droite à un σ à 5.0.



Grille avec σ à 0,0001



Grille avec σ à 5

Pour un σ à 0,0001, l'aire de la grille est de 1,830 et l'erreur de quantification vectorielle moyenne est de 0,0084.

Pour un σ à 5, l'aire de la grille est de 0,1211 et l'erreur de quantification vectorielle moyenne est de 0,0517.

On constate qu'avec un petit σ, la grille est encore plus désorganisée que ce nous avions pu constater précédemment et prend beaucoup de place. Ceci est du au fait que très peu de neurones se déplacent à chaque nouvelle entrée.

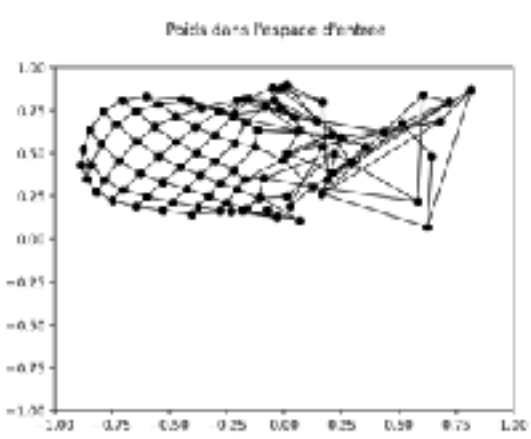
Parallèlement, avec un σ à 5, comme annoncé au préalable, la grille est très serrée et bouge uniformément entre chaque affichage. Beaucoup de voisins apprennent de chaque entrée entrainant ainsi un déplacement de chaque apprenant.

Analyse du nombre de pas de temps d'apprentissage N

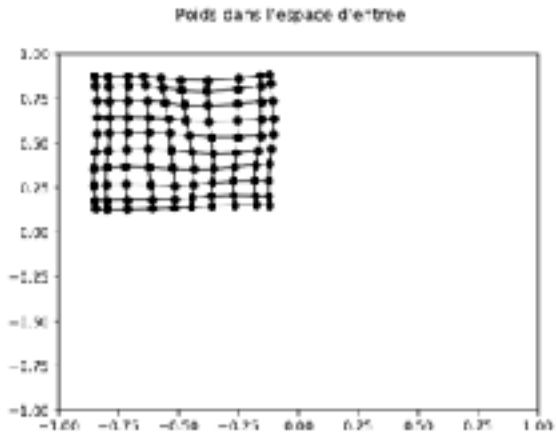
Pour l'analyse du nombre de pas de temps d'apprentissage N, j'ai réalisé deux simulations avec les autres paramètres aux valeurs initiales. Sur l'image de gauche, il y a eu 1000 pas de temps alors que sur l'image de droite, il y en a eu 60000.

Pour un N à 1000, l'aire de la grille est de 1,4121 et l'erreur de quantification vectorielle moyenne est de 0,0091.

Pour un N à 60000, l'aire de la grille est de 0,5727 et l'erreur de quantification vectorielle moyenne est de 0.0043.



Grille avec N à 1000



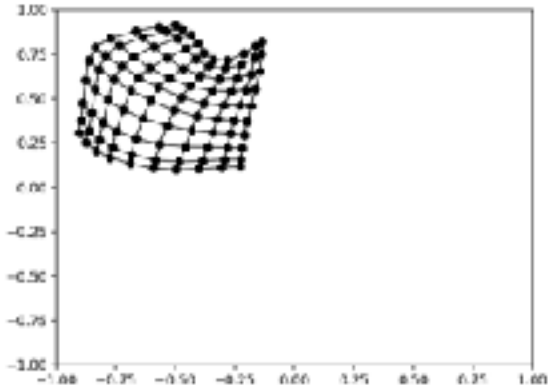
Grille avec N à 60000

Pour N à 1000, on peut aisément imaginer que la carte n'a pas eu le temps de se mettre complètement en place. Evidement, ceci est dû au fait que le nombre de pas de temps est trop petit.

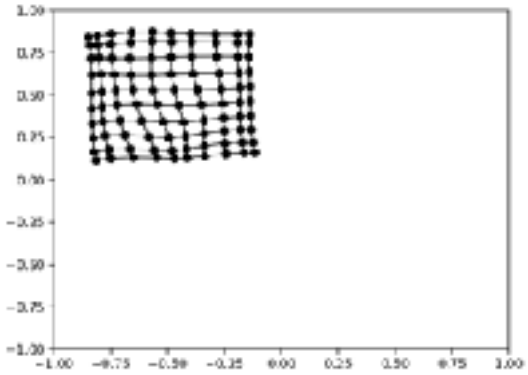
Pour N à 60000, on constate que la grille est bien en place. L'aire et l'erreur de quantification vectorielle moyenne sont semblable à la situation initiale. On peut donc supposer que le nombre de pas est trop grand et que 30000 pas de temps sont suffisants.

J'ai cherché à réduire la valeur de N de façon à trouver le N le plus petit qui permet d'obtenir une grille. Avec un N = 5500, nous avons une aire à 0,6014 et une erreur de quantification vectorielle à 0,0077. La grille n'est pas encore formée. Avec un N = 6000, l'aire de la grille est de 0,5783 et l'erreur de quantification vectorielle est à 0,0044. La grille est maintenant formée.

Nous pouvons en déduire que 6000 pas de temps suffisent pour former la grille. L'aire de la grille est proche de l'aire initiale (0,5783 à N = 6000 et 0,5653 pour N = 30000) et l'erreur de quantification vectorielle est presque qu'identique. Pour un résultat plus rapide, 6000 pas de temps semble être une valeur suffisante dans ce contexte.



Grille avec N à 5500



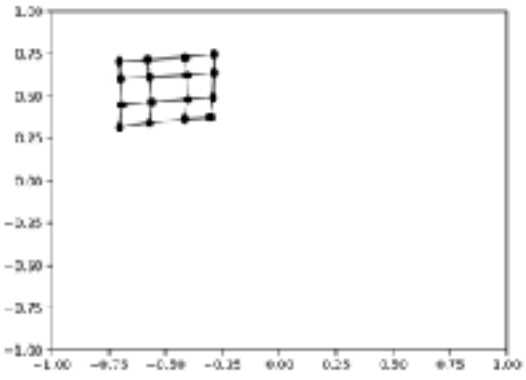
Grille avec N à 6000

Taille et forme de la carte

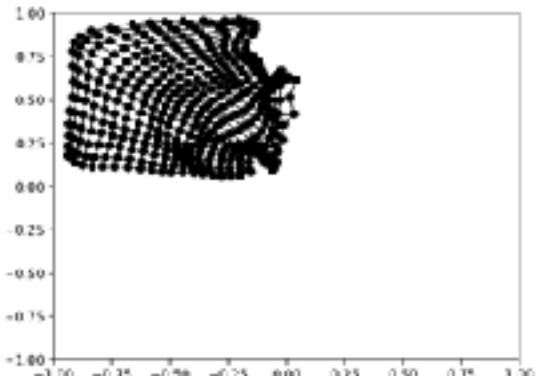
Dans un premier temps, je vais changer la taille de la carte. En réinitialisant les paramètres et en utilisant une grille 4x4, nous obtenons une grille avec une surface de 0,1791 et une erreur de quantification moyenne de 0,03989. On constate que l'aire est plus petite et que l'erreur de quantification est très nettement supérieur comparé à une grille 10x10.

Dans le cadre d'une grille 20x20, l'aire de la grille est de 0,8850 et l'erreur de quantification est de 0,0012. Il est important de signalé que le processus est, logiquement, beaucoup plus long.

Pour rappel, à l'origine nous avions une aire de 0,5653 et une erreur de quantification vectorielle de 0,0045. Notre grille est donc plus grande et avec moins d'erreurs. En revanche, comme vous pouvez le constater sur la capture ci-dessous elle est également moins bien formée. Nous pouvons en conclure que plus nous avons de neurones plus nous sommes précis. En revanche, avoir trop de neurones s'avère être un handicape en terme de performance et d'efficacité.



Grille de 4 x 4

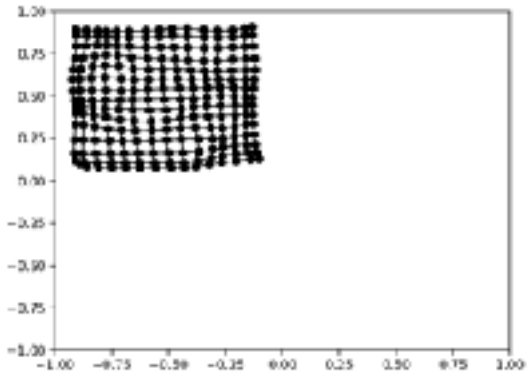


Grille de 20 x 20

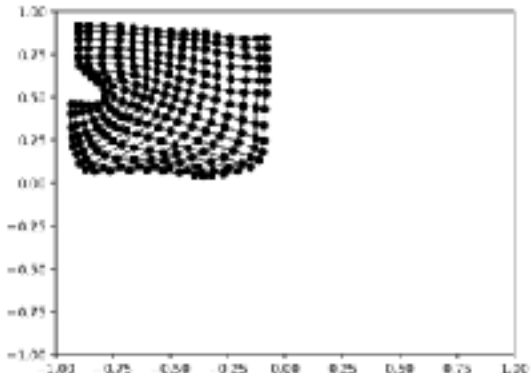
Je vais donc chercher qu'elle est la taille maximale où nous conservons une grille correctement structurée. Ainsi, nous gagnerons en précision en réduisant l'erreur tout en ayant une structure pertinente.

Il semblerait que la bonne taille soit 15*15. Nous obtenons une aire de 0,6824 et une erreur de quantification vectorielle moyenne à 0,0018. Comme vous pouvez le constater, la grille est toujours bien formée. Nous obtenons donc une aire plus grande qu'avec une grille 10*10 et une erreur de quantification vectorielle moyenne moins importante.

J'ai pris le temps d'ajouter le rendu de la grille 16*16 afin de montrer la déstructuration de celle-ci.

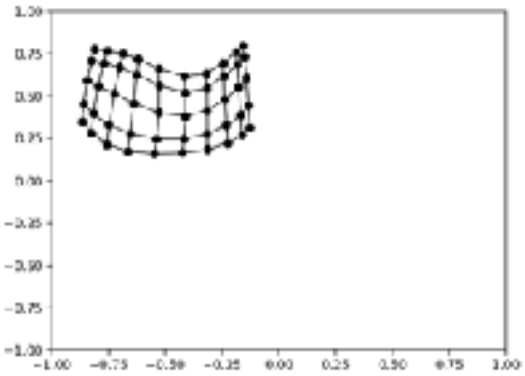


Grille de 15 x 15



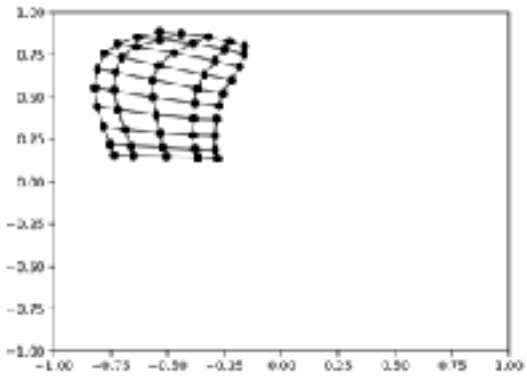
Grille de 16 x 16

J'ai réalisé une simulation avec une carte 5 x 10. On constate aisément que la grille subit des déformations et n'est pas aussi structurée qu'avec les grilles carrées inférieur à 16. L'aire de la grille est de 0,4708 tant dit que son erreur de quantification vectorielle moyenne est de 0,0125.



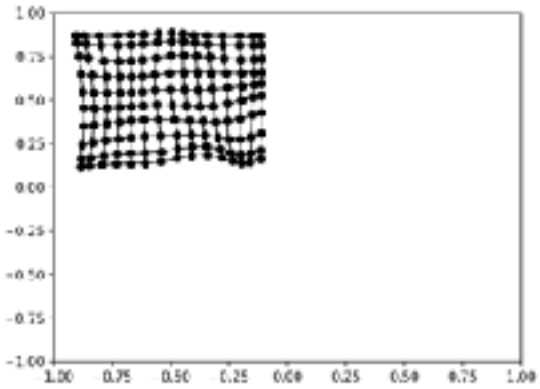
Grille 5 x 10

J'ai effectué plusieurs simulations avec une carte 5 x 10 et j'ai constaté que parfois, la grille pouvait pivoter de manière à devenir comme ci-dessous. Je suppose que c'est dû à l'initialisation des poids au départ.



Grille 5 x 10 inversée

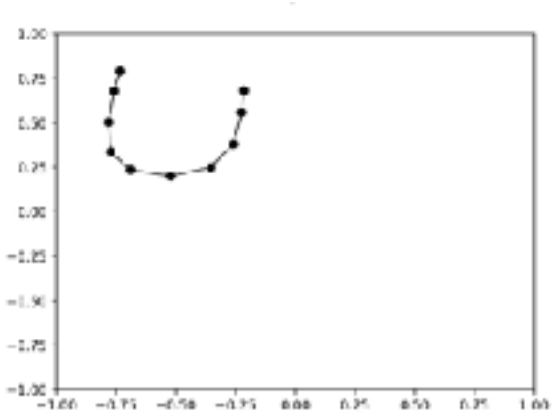
J'ai réalisé une simulation avec une grille rectangulaire de taille plus importante. On peut constater sur la capture ci-dessous qu'avec une grille 10 x 15, la grille semble plus structurée dans ces conditions. Le taux d'erreur de quantification vectorielle moyenne est à 0.0032.



Grille 10 x 15

J'ai lancé une simulation sur une grille 12 x 12. L'objectif est de comparer le taux d'erreur de quantification vectorielle moyenne pour deux cartes avec un nombre de neurones proche (150 et 144). Pour une grille 12 x 12 ce taux est de 0.0028. On peut donc en déduire qu'il est plus interessant d'utiliser une grille carrée plutôt que rectangulaire. Nous somme également gagnant en ressource car ici nous utilisons 6 neurones de moins.

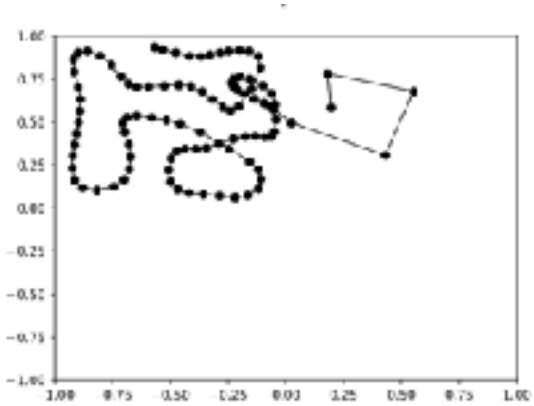
Enfin, pour conclure sur la forme de la carte, j'ai testé avec une ligne de 10 neurones à plusieurs reprises. Je m'attendais à obtenir une droite en diagonale mais j'ai obtenu le résultat suivant.



Exemple de ligne de 10 neurones

Après réflexion, j'ai réalisé que le résultat était logique, il n'y avait aucune raison pour qu'aléatoirement les neurones aux extrémités soit prédestinés à se rendre dans les coins plutôt qu'à un autre endroit. L'erreur de quantification vectorielle moyenne est de 0,0350 sur les trois simulations.

Suite à cela, j'ai lancé une simulation avec une lignes de 100 neurones afin de voir si c'était une alternative viable à la grille 10 x 10. Le rendu est affiché ci-dessous. Avec cette configuration, j'obtiens un taux d'erreur de 0,0035. Ma ligne de 100 neurones loge dans un rectangle d'une aire de 1,286.

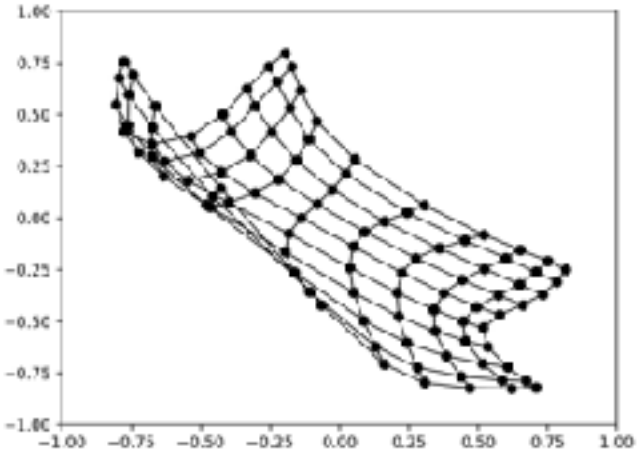


Ligne de 100 neurones

Si l'on compare avec les valeurs initiales, le taux d'erreur est semblable. En revanche, l'aire est plus importante avec la ligne de 100 neurones. On constate graphiquement que parmi les 100 neurones, une partie n'est pas dans le cadre du jeu de donnée, augmentant ainsi l'aire. Il semble donc plus efficace d'utiliser une grille 10 x 10 plutôt qu'une ligne de 100 neurones pour couvrir le jeu de donnée

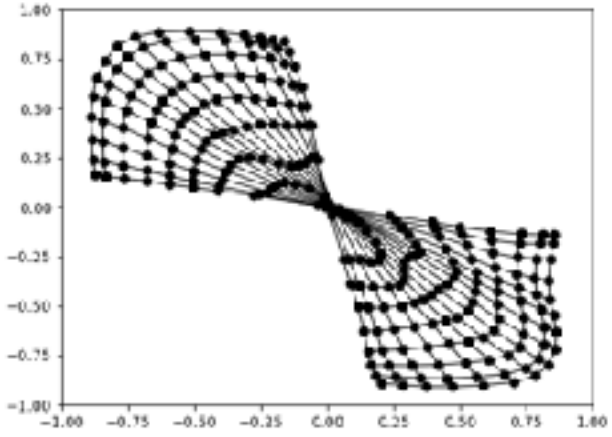
Analyse du jeu de données

Dans un premier temps, j'ai uniquement changé le jeu de données pour le jeu numéro 3, tout en gardant les paramètres initiaux. On constate aisément sur la capture ci-dessous que le qualitativement, nous ne somme pas très bon.



Paramètre initiaux avec le jeu de données n°3

Dans un second temps j'ai cherché les meilleurs paramètres pour ce jeu de données. Le meilleur rendu semble être avec une grille 16 x 16.

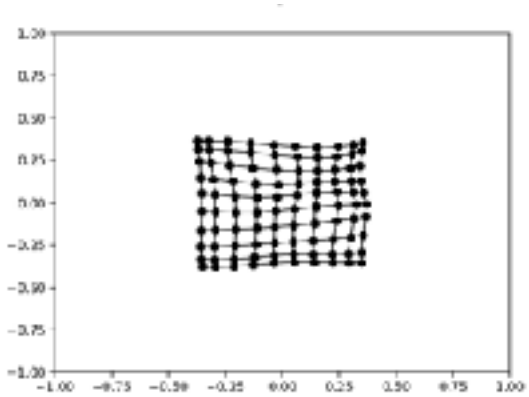


Jeu numéro 3 et grille 16 x 16

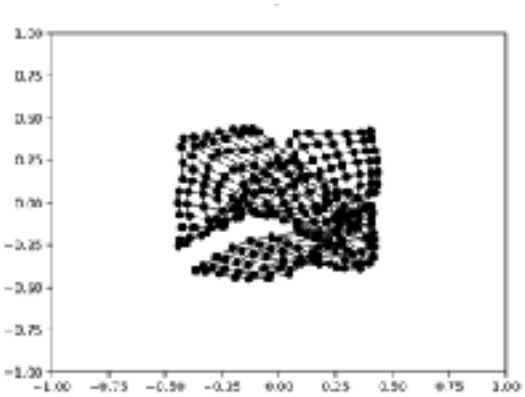
J'ai tenté de modifier les autres paramètres afin d'obtenir de meilleures statistiques, mais sans succès. Avec les paramètres initiaux et une grille 16 x 16, nous obtenons donc une aire de 3,1003 et une erreur de quantification vectorielle moyenne de 0,0040.

Après cet essai, j'ai voulu expérimenter ce qui se passait si les neurones étaient placés au même endroit que les données. J'ai donc tout déplacé entre -0,50 et 0,50. Après plusieurs

essais avec différents paramètres j'ai constaté que la grille allait se placer correctement en 11500 pas de temps avant de se désorganiser à 30 000 et plus.



Grille centrée et N = 11500

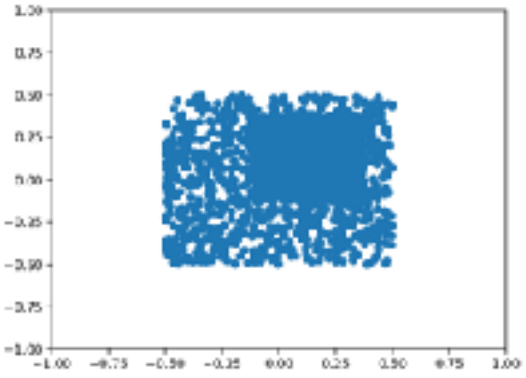


Grille centrée et N = 60000

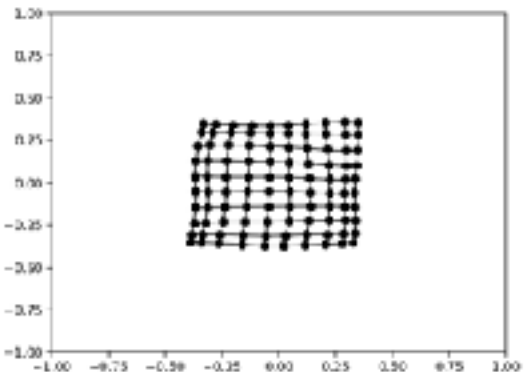
L'aire de la grille est de 0,5531 et l'erreur de quantification vectorielle moyenne est de 0,0047.

Nous avons vu précédemment, que nous pouvions réduire à 15000 pas de temps lorsque que les données et la carte étaient sur deux zones différentes. Ici, nous pouvons voir qu'avec 11500 pas de temps, nous sommes à un résultat équivalent. Avec un placement différent nous arrivons donc à obtenir de meilleures performances.

J'ai rajouté une zone danse à l'intérieur d'un carré comme sur l'image ci-dessous à gauche. Le but était de voir comment les neurones allait se placer et voir si la grille allait être chamboulée. Le résultat est présent à ses côtés.



Données avec une zone plus dense



Résultat en fonction des données

On peut constater qu'avec les paramètres d'apprentissages actuels, il n'y a pas de changements majeurs à avoir une zone plus dense à l'intérieur des données.

Bras robotique

Une fois la carte apprise, avec les positions motrices θ1 et θ2 il est possible d'anticiper la position du bras à l'aide des valeurs indiqué par les positions motrices. Les angles étant singuliers et compris entre 0° et 360°, il existe qu'une unique position du bras.

$x1' = 0 + l1 * \cos(\theta 1)$
$x2' = 0 + l1 * \sin(\theta 1)$

$x1 = x1' + l2 * \cos(\theta 2)$
$x2 = x2' + l2 * \sin(\theta 2)$
Nous obtenons donc les composantes de la position (x1,x2).

Dans la situation inverse, si l'ont souhaite atteindre une position spatiale, il est nécéssaire de calculer les positions motrices (θ1, θ2) de manière à trouver à la position en x1 et x2, souhaitée. En effet, il existe plusieurs paires d'angles (θ1, θ2) pour retrouver les positions x1 et x2.

L'autre modèle vu dans les vidéos du cours est le modèle du gaz neuronal. L'avantage d'utiliser ce modèle est qu'il ne prend pas en compte la forme de la carte. Cet aspect est intéressant dans le cadre des données de la position de la main qui ne forment pas une forme triviale. L'inconvénient de ce modèle est qu'il prend en compte toutes les données. De ce fait, ce qui n'est pas forcément le cas ici, il est sensible au bruit. En plus de cela, si l'espace d'entré est grand, ce modèle nécessite beaucoup de neurones.

Dans le cas où l'objectif serait de simplement prédire la position spatiale à partir de la position motrice, il aurait été opportun d'utiliser le modèle du Perceptron. Il procure l'avantage d'obtenir une convergence plus rapide mais surtout il nous garanti l'existence de celle-ci. L'inconvénient de ce modèle est qu'il fonctionne uniquement avec des séparations linéaires.

Pour connaitre la position de la main d'une position (θ1, θ2) à une positions (θ'1, θ'2), il suffit de tracer le rectangle avec pour l'un des angles la position initiale et pour l'angle opposé la position finale. Ainsi, comme il existe plusieurs façons de rejoindre la position finale - donc, plusieurs trajectoires - nous auront toutes les positions spatiales susceptibles d'être traversées.