

Dpt Informatique

Rapport Techniques d'Intelligence Artificielle Carte de kohonen

Mohamed El Mehdi MAKHLOUF p1808420
SALMI DORIAN p1710287

Encadré par Monsieur LEFORT MATHIEU

Année universitaire 2020/2021

3) Étude “théorique” des cas simples

3.1/ Influence de η

- Si $\eta = 0$, la variation des poids est 0, et donc les poids restent constantes W^*
- Si $\eta = 1$, la variation des poids est $(X - W^*)$, donc les poids du neurone deviennent $W^* + (X - W^*)$, donc la valeur des poids du neurones est égale à X avec X notre entrée.
- Si η est dans $]0; 1[$, alors la variation est $\eta(X - W^*)$

On peut borner cette valeur par les deux valeurs trouvées précédemment

$$\begin{aligned} 0 &\leq \eta(X - W^*) \leq X \\ \Rightarrow W^* &\leq W^* + \eta(X - W^*) \leq W^* + X \end{aligned}$$

Le nouveau poids peut s'écrire : $(1 - \eta)W^* + \eta X$, ce qui s'apparente à une somme pondérée

Donc notre nouveau vecteur des Poids du neurone gagnant sera forcément plus grand ou égale à vecteur du poids actuel si $\eta = 0$ et il ne peut pas dépasser la somme de sa valeur actuelle et l'entrée ($W^* + X$).

3.2/ Influence de σ

- Si σ augmente, les neurones proches du neurone gagnant vont apprendre plus de l'entrée courante, en addition on aura plus de neurones qui vont apprendre de notre entrée.

-Mathématiquement : notre fonction de voisinage est une fonction gaussienne qui dépend de l'écart type (ou sigma : largeur du voisinage gaussienne).

-De cette manière, on essaie de regarder l'influence de sigma sur la variation des poids pour les neurones les plus proches du neurone gagnant. On remarque que si le sigma augmente, la distance au carré divisée par sigma au carré va décroître et du coup l'exponentiel de la négation de cette dernière valeur (distance au carré divisée par sigma) va croître. Autrement, si on augmente notre sigma, les neurones les plus proches de notre neurone gagnant vont apprendre plus.

- En conclusion, si σ est plus grand, après la convergence l'auto-organisation obtenue va être plus resserrée. Logiquement, si les neurones les plus proches apprennent plus de notre entrée, ils seront plus resserrés. Donc, vu qu'on est en train d'influencer plus les neurones avec un sigma plus grand (un diamètre d'influence plus grande), forcément on va finir avec une carte plus resserrée.

- Pour évaluer l'auto-organisation de notre carte, on va mesurer la plus grande distance entre deux vecteurs de poids des neurones. Si on veut appeler cette mesure autrement, c'est le diamètre de notre carte une fois qu'elle a fini de s'auto-organiser, plus la métrique est proche de 0 plus l'auto-organisation est "resserrée" et plus la métrique est grande plus l'auto-organisation est "lâche".

3.3/ Influence de la distribution d'entrée

Q1.1/

- Si on présente à notre neurone autant de fois deux différentes entrées (X1 et X2) avec un taux d'apprentissage faible et on répète le processus suffisamment des fois, le vecteur du poids W de notre neurone va tendre **vers le milieu entre les deux entrées X1 et X2**.
- Si on va lui présenter une entrée X1, il va se rapprocher un peu vers cette entrée et si on lui présente la deuxième X2, il va se rapprocher un peu vers cette dernière à chaque fois. A fin le vecteur des poids de notre neurone W convergera vers :

$$W = \frac{X1+X2}{2}$$

Q1.2/

- Si on refait la même procédure en présentant X1 N fois de plus par rapport à X2 le **vecteur de poids de notre neurone va se rapprocher plus vers l'entrée X1** vu qu' elle sera présentée plus de fois par rapport à l'entrée X2.
- Le vecteur de poids de notre neurone va finir proportionnellement plus proche de X1 en fonction de n , Si notre n est suffisamment grand et on répète la procédure plusieurs fois le vecteur de poids va finir vers l'entrée X1.
- Si on veut se baser sur les formules on peut estimer que le vecteur des poids de notre neurones convergera finalement vers :

$$W = \frac{N*X1+X2}{N+1}$$

Q2/

- Les vecteurs des poids de nos neurones vont se rapprocher plus vers les zones où la densité des données est plus élevée. Dans les zones où la densité des données est petite, on va trouver peu de neurones finalement et c'est typiquement le cas pour

les vecteurs d'entrée rare. Finalement, on peut dire que **la répartition des neurones est calquée sur la densité des données**.

4) Étude pratique

4.3\ Analyse de l'algorithme

Dans les réseaux de neurones, ils existent différents hyper paramètres que l'on peut modifier pour avoir les meilleures performances, ainsi le but de cette partie est d'analyser l'influence des différents hyper paramètres sur les performances du réseau mesurable notamment grâce à l'erreur de quantification vectorielle **MSE** et la mesure d'auto organisation proposée dans la partie **3.2** qu'on va appeler dans la suite **MAO**.

Taux d'apprentissage η

Étudions dans cette partie l'influence du taux d'apprentissage sur la performance de notre réseau, J'ai pu effectuer des mesures de la MSE et MAO en fonction du taux d'apprentissage sur le premier jeu de données en regroupant les mesures dans la figure ci-dessous.

η	0	0.01	0.05	0.3	0.7	0.9	1.0
MSE	0.37	0.026	0.024	0.021	0.023	0.023	0.021
MAO	1.24	1.94	1.96	1.97	2.10	2.023	2.16

Figure 1 – Évolution du MSE, MAO de η (Jeu de données n°1)

En analysant les valeurs de la Figure 1 on remarque de légères fluctuations sur la MSE et la MAO à partir d'un taux d'apprentissage supérieur ou égale à 0,01. On peut considérer ces fluctuations comme du bruit et qui peuvent venir aussi du tirage aléatoire des données. Mais on voit quand même que choisir un taux d'apprentissage plus grande ne donne pas forcément des meilleurs résultats

Nous allons refaire la même procédure pour le 3ème jeu de données pour voir si on va remarquer la même chose ou pas.

η	0	0.001	0.01	0.05	0.07	0.1	0.3	0.5	0.7	0.9	1.0
MSE	0.38	0.025	0.025	0.013	0.013	0.015	0.016	0.012	0.011	0.017	0.016
MAO	1.20	1.94	1.94	2.13	2.17	2.08	2.03	2.15	2.37	2.07	2.24

Figure 2 – Évolution du MSE, MAO en fonction de η (Jeu de données n°3)

- Grâce aux mesures effectuées, on peut remarquer que si le taux d'apprentissage est trop petit par rapport à un taux d'apprentissage seuil, notre carte sera mieux organisée à condition de mettre plus de pas d'apprentissage (une remarque confirmée un peu plus bas), pour donner suffisamment de temps aux notre carte de s'auto organiser et converger. Dans le cas où on met un taux d'apprentissage petit qu'un taux seuil avec peu d'itération les neurones de notre carte vont peu apprendre et risquent de finir loin du point de convergence et donc on aura des mauvaises performances.
- On peut aussi dire d'après nos mesures que à partir d'une valeur seuil de η , l'augmentation de ce dernier n'améliore pas nos performances.
- Pour conclure cette partie, on est certain que la valeur du taux d'apprentissage influence les performances de notre carte et qu'il faut essayer plusieurs valeurs pour trouver le seuil où on a les meilleures performances.

Largeur du voisinage σ

Pour étudier l'influence du longueur de voisinage, nous avons lancé l'entraînement de la carte sur le même jeu de données plusieurs fois en faisant varier la longueur de voisinage σ et en prenant des mesures d'auto organisation MAO, afin de vérifier notre hypothèse émise dans la partie 3.2 que plus sigma est grand plus l'auto organisation serait resserrée.

σ	0.1	0.7	1.4	2.8	5	10
MSE	0.014	0.011	0.027	0.071	0.21	0.49
MAO	2.30	2.15	1.94	1.53	0.99	0.28

Figure 3 – Évolution du MSE, MAO en fonction de σ (Jeu de données n°1)

- Nous avons pu confirmer, grâce à notre mesure d'auto d'organisation de notre carte (la plus grande distance entre deux poids de notre carte), que notre carte est effectivement plus resserrée si on augmente la largeur du voisinage. (La valeur de notre mesure diminue en augmentant la largeur de voisinage ce qui signifie que la carte est plus resserrée).
- On remarque aussi que notre carte est plus relâchée quand on diminue la largeur de voisinage.
- Ce qu'on peut constater également c'est que si on augmente notre largeur de voisinage, les poids des neurones de la carte vont se rapprocher plus vers l'endroit où on a plus de données d'entrées et donc elles s'éloignent plus des données rares. Cela peut influencer l'erreur de quantification vectorielle moyenne légèrement, vu que notre erreur peut être plus grande juste sur les données rares et on assure une bonne erreur sur les données les plus fréquentes. Mais de manière générale choisir

une grande largeur de voisinage peut influencer les performances de la carte surtout sur les données les plus rares.

Nombre de pas de temps d'apprentissage N

N	3	30	300	3000	30000	300000
MSE	0.35	0.26	0.077	0.031	0.018	0.026
MAO	1.27	1.49	1.69	1.88	2.06	1.91

Figure 4 – Évolution du MSE, MAO en fonction de N (Jeu de données n°1)

Après plusieurs exécutions sur le même jeu de données avec différents nombres de pas d'apprentissage, nous remarquons que:

- Pour avoir une meilleur erreur de quantification vectorielle il faut trouver le bon nombre de pas d'apprentissage adéquat avec le taux d'apprentissage et même la taille de notre jeu de données en entrée. (Le nombre de pas d'apprentissage doit être plus grande que la taille de nos données en entrée si on veut que notre carte apprend sur toutes les entrées)
- Si on dépasse ce nombre de pas d'apprentissage, notre carte va sur apprendre et les poids de nos neurones vont se rapprocher plus vers les entrées les plus présentées, ce qui fait augmenter l'erreur de quantification vectorielle.(ça ressemble un peu au phénomène crée quand on choisit une largeur de voisinage élevé)
- Dans le même sens, si on choisit un nombre de pas plus petit que le nombre de pas optimum, les poids de notre carte vont se rapprocher vers le peu d'entre présentés et donc on a une erreur de quantification vectoriel élevée. La carte n'a pas été suffisamment entraînée sur toutes les entrées .
- Pour la mesure d'auto-organisation qu'on a proposé, on peut remarquer qu'on a la meilleure mesure pour un nombre de pas d'apprentissage optimal. Si on entraîne notre carte sur un nombre de pas d'apprentissage inférieur au nombre de pas optimal, la mesure va être inférieure. Les poids de notre neurone vont être recentrés sur les premières entrées présentées à notre carte.
- Si on choisit un nombre de pas plus grand que le nombre de pas optimum, la carte va sur apprendre et les poids des neurones de la carte vont se rapprocher plus vers les entrées les plus présentées et donc notre mesure d'auto-organisation va être légèrement inférieure à la mesure optimale.
- On en conclut alors facilement que le surentraînement n'apporte rien en plus et très certainement qu'en fonction de nos données il vaut mieux trouver le nombre N idéal plutôt que celui-ci soit trop élevé. En trouvant le N idéale à notre et notre taux d'apprentissage et à notre jeu de données, on peut grandement gagner en temps d'apprentissage.

Taille et forme de la carte:

Nous allons tester différentes formes de cartes pour le même jeu de données pour voir l'influence de la forme de la carte sur nos résultats.

Forme de la carte	(1*100)	(50*2)	(2*50)	(20*5)
MSE	0.013	0.024	0.024	0.022
MAO	2.29	2.09	2.09	1.94

Figure 5 – Évolution du MSE, MAO en fonction de la taille de la forme de la carte (Jeu de données n°1)

- D'après les mesures, on peut constater que les performances d'une carte de taille (A*B) est presque la même qu'une carte de taille (B*A). On peut voir cela mathématiquement aussi, vu que les positions des neurones interviennent dans le calcul d'une distance et prendre la forme transposée de la carte n'influence pas les distances et donc ça ne change rien dans nos calculs.
- Pour une carte sous forme de ligne ou colonnes, peu de neurones vont apprendre à chaque fois par rapport à une autre forme (par exemple, matricielle) ce qui va nous conduire vers une carte plus relâché et donc une mesure d'auto organisation plus élevée et une meilleure erreur de quantification vectorielle cela vient du fait que en prenant une carte de base plus relâché ou à chaque fois qu'on a un neurone gagnant le nombre de neurone dans le diamètre de la largeur de voisinage est plus petit par rapport à une autre forme de carte.
- On peut remarquer également qu'une carte avec un nombre plus grand de neurones va mieux s'auto organiser, c'est-à-dire elle sera plus resserrée à la fin et donnera de meilleurs résultats en terme d'erreur de quantification.
- Pour conclure cette partie, la taille et la forme de la carte va influencer la l'interaction entre les neurones, et plus précisément le phénomène d'attraction entre les neurones et le neurone gagnant, et il vaut mieux privilégier une certaine forme de carte selon la distribution de nos données.

Jeu de données

Nous allons tester notre carte sur les trois différents jeu de données avec les mêmes hyper paramètres ($\eta=0.005$, $\sigma=1.4$, $N=30000$ et une carte de taille (10*10)) pour voir l'influence du jeu de données sur nos résultats.

Jeu de donnée	1 ère	2ème	3ème
MSE	0.017	0.18	0.12
MAO	2.04	2.04	2.01

Figure 6 – Évolution du MSE, MAO en fonction du jeu de donnée

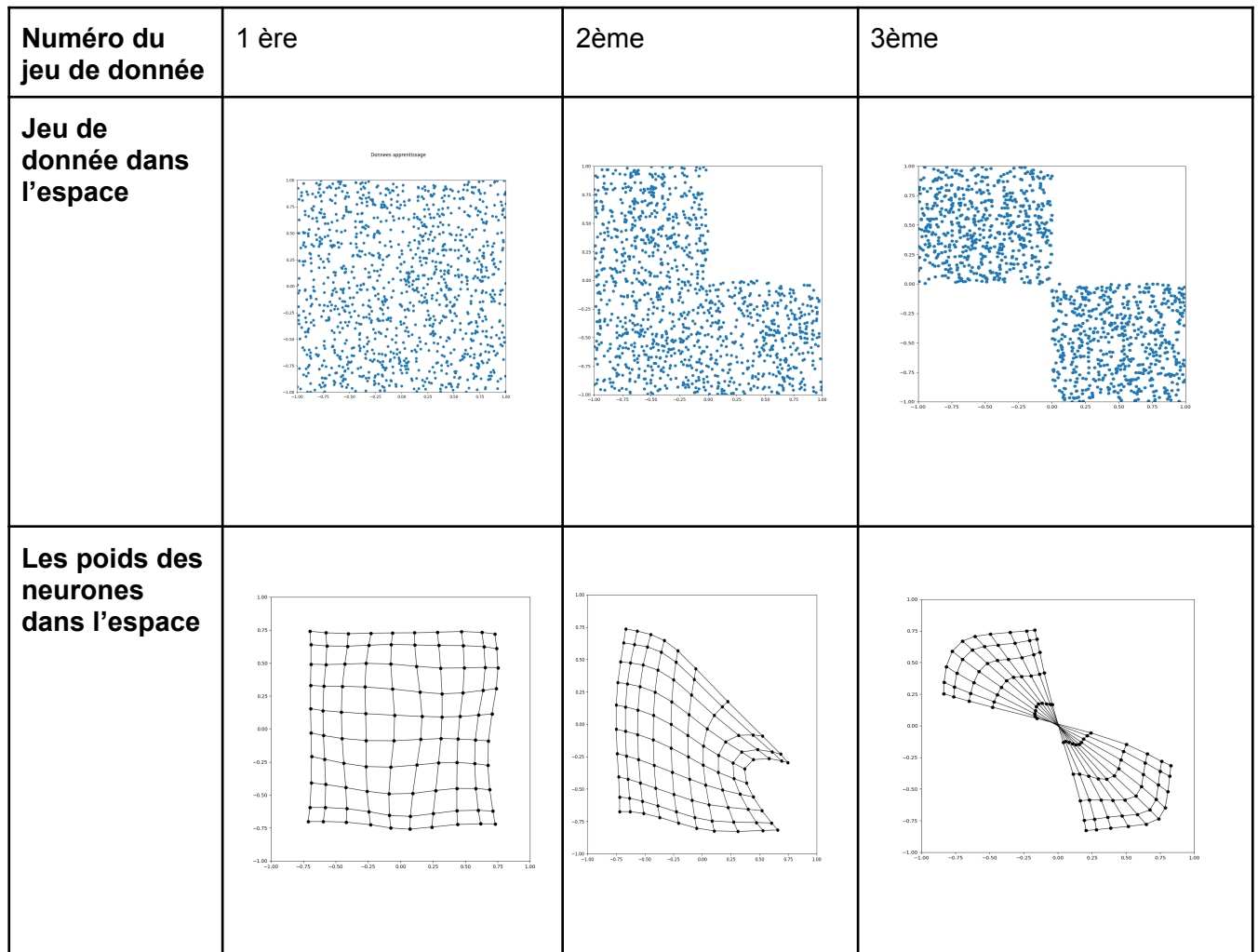


Figure 7 – Distribution des poids des neurones en fonction du jeu de donnée

- En testant notre carte sur les 3 différents jeux de données proposés dans le sujet, et en observant la répartition des poids dans l'espace, nous avons pu confirmer que les poids de nos neurones se répartissent dans l'espace selon la distribution des données d'entrées cf figure 7.
- Nous remarquons également des meilleures performances en terme d'erreur de quantification vectorielle moyenne ou bien en terme d'évaluation d'auto-organisation de la carte sur le troisième jeu de données par rapport aux deux autres jeux de

données pour un entraînement avec les mêmes hyper paramètres, ce qu'il veut dire qu'il faut adapter les paramètres selon le jeu de données pour atteindre les même performances.

- On a pu confirmer cela en créant un jeu de données non uniforme. On a pu remarquer qu'on a plus de neurones dans la partie de l'espace où on trouve plus de données, veuillez trouvez ci dessous dans la **figure 8** les résultats de l'entraînement de la carte avec les mêmes hypers paramètres utilisés sur les autres jeux de données ($\eta=0.005$, $\sigma=1.4$, $N=30000$ et une carte de taille $(10*10)$).

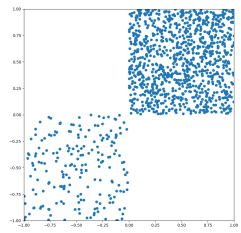
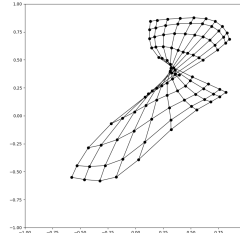
MSE	MAO	Jeu de donnée en entré	Carte après organisation
0.017	1.19		

Figure 8 – Performances de la carte et la distribution de ses poids sur notre jeu de donnée

Conclusion de l'analyse

On peut conclure que chaque hyper paramètre du réseau influence plus ou moins sur les performances de la carte, mais qu'il existe surtout des paramètres "optimus" qui permettent notamment de gagner en temps d'entraînement et en MSE, il ne suffit pas seulement de mettre les plus grandes valeurs à chaque fois, pour un jeu de données il existe des paramètres optimums et une configuration réfléchie performante bien mieux qu'avec de grandes valeurs sans sens.

4.4 Bras robotique

Question 01 :

- Une fois que le carte a fini d'apprendre, on peut passer d'une coordonné spatiale à une coordonné motrice juste en donnant en entrée à notre réseau les coordonnées spatiales. On regarde le neurone gagnant par rapport à ce vecteur d'entrée de taille 2 et on va s'intéresser à son vecteur de poids qui fait une taille (4*1). Dans ce cas là, les cordonnées motrices approximatives correspondantes aux coordonnées spatiales données en entrée vont être la 3ème et la 4ème composante de notre vecteur de poids.
- Pour passer d'une coordonnée motrice a une coordonnée spatiale, on peut procéder de la même manière, c'est à dire on donne en entrée de notre carte un vecteur de taille de deux qui correspond à notre coordonnée motrice et on va chercher le neurone gagnant parmi tous les neurones on comparant la coordonnée donnée en entrée à la bon partie des poids des neurones qui correspond aux coordonnées motrices. Une fois qu'on a réussi à sélectionner un neurone gagnant, on va s'intéresser aux deux composantes restantes de son vecteur du poids, qui vont représenter en fait la coordonné spatiale approximative correspondante à la coordonné motrice donnée en entrée.
- Pour l'implemenation, vous trouverez sur le fichier Kohnen.py déposé en pièce jointe a ce rapport une fonction **SpatialToMotriceORInversse** qui peut faire le passage dans les deux sens. Il faut lui passer en addition au couplet composant la position un booléen à true si on veut passer d'une coordonné motrice a une coordonnée spatiale.
- Le principe a été choisi en se basant sur le fait que les poids de nos neurones vont se rapprocher vers nos donnés. Du coup, pour estimer une partie des données, on peut essayer de trouver le vecteur des poids le plus proche de la partie connue de nos données et puis on remplacera la partie manquante par les poids du neurone le plus proche.
- Une autre implémentation possible serait de prendre en compte la distance entre les coordonnés du neuron gagnant et les coordonnées donnés en entrée. En fonction de cette différence, si elle est pas nulle, c'est a dire si l'entré ne correspond pas exactement à une partie d'un vecteur de poids d'un neurone, on va s'intéresser plutôt dans ce cas là aux quatre neurones qui entourent notre entrée. (avec un effet miroire si le neurone gagnant se trouve sur les extrémités de la carte (premier /dernière) (ligne /colonne)). On va essayer de faire une moyenne pondérée en fonction de la distance entre chaque neurones entourent notre entrée et les poids de ces derniers en suivant la formule suivant:

$$Coordonné = \frac{\sum V_i^* \frac{1}{DXVi}}{\sum \frac{1}{DXVi}}$$

V_i : La partie qui nous intéresse du vecteur des poids du i ème voisin. ($i=1, \dots, 4$)

DX_i : La distance entre la bonne partie* du i ème voisin et la coordonnée donnée en entrée.

La bonne partie* : Ça dépend quelle coordonnée on veut estimer.

- Cette méthode aussi reste toujours approximative, mais elle prend en compte la différence entre l'entrée et le meilleur neurone en essayant d'estimer la position recherchée en exploitant cette distance donc avec cette méthode on va essayer de minimiser les erreurs d'approximation.
- Vous trouverez dans le code l'implémentation de cette deuxième méthode d'approximation dans une fonction appelée **SpatialToMotriceORInverseMoyenne** qui fonctionne pour les estimation dans les deux sens avec le même système de booléen passée en paramètre pour faire l'approximation dans les deux sens .

Question 02:

- La méthode d'apprentissage sur le quadruplet pour les mémoriser afin de retrouver une sous partie ressemble au principe des réseau de neurone de **Hopfield** qui a un inconvénient bien connu : la capacité de stockage qui correspond à 0.14 nombre de neurones motifs. Donc si on veut stocker un grand nombre de quadruplets, on aura besoin d'un nombre immense de neurones et aussi le fait qu'on va apprendre par cœur notre jeu de données avec et même les données bruitées seront inclus.
- L'avantage de ce modèle est qu'on va apprendre par cœur les quadruplets des positions, donc on n'aura pas d'erreur sur notre prédiction.
- Pour éviter la limitation par rapport au nombre de neurones, Il existe un autre model qui peut être plus intéressant qu'un Hopfield et qui fonctionnera pour la tâche de prédiction dans les deux sens et qui marche presque comme la carte de kohonen pour son apprentissage et même avec la prédiction: c'est le **Gaz Neuronal**. La limitation de ce modèle premièrement c'est que on va apprendre les données avec le bruit (typiquement pour notre cas avec les données du bras du robot).
- En addition on a toujours besoin de beaucoup de neurones si la taille de nos données est grande mais on aura besoin de moins de neurones que ce qu'on avait besoin pour un Hopfield.
- Donc en bref , un bon remplaçant de notre carte de Kohonen pour faire la prédiction des coordonnées dans les deux sens ça serait un **Gaz Neuronal**.

Question 03:

- Pour prédire uniquement la position spatiale, on peut utiliser à la place de notre carte une Perceptron multicouches, qui représente un approximateur universelle qui peut nous garantir des positions spatiales en sortie du réseau avec un taux d'erreur et comme dit dans le cours sa convergence est garantie mais pas absolue.
- Ce modèle est limité par sa convergence peu efficace. On risque d'avoir besoin de plusieurs couches cachées et donc plusieurs variables de poids pour avoir un bon taux d'erreur. On n'oublie pas également de mentionner que le paramétrage du réseau peut prendre beaucoup de temps, il faut faire de l'optimisation à la main pour trouver le bon nombre de couches et le nombre de neurones par chaque couche et pour choisir les bon paramètres de l'algorithme.
- Un autre avantage pour ce model c'est que nos neurones dans ce cas la il vont pas essayé de se répartir sur le jeu d'entré ce qui peut être difficile à faire pour des donnés réparti d'une façon un peu complexe (comme on peut le voir sur les donnés de notre bras du robot) mais plutôt il vont essayer de les séparer grâce à des plans dans notre cas et c'est le principe de la régression linéaire qui nous intéresse ici et qui peut nous garantir des bonne performances à condition de bien paramétrer notre réseau de neurones.
- Une simple perceptron aussi peut marché à la place de la perceptron multi couche, j'ai préféré d'utiliser une perception multi couches même si elle est plus coûteuse en termes de ressources de calcule pour assurer des meilleurs performances

Question 04:

- Pour tracer grossièrement la trajectoire spatiale du bras du robot d'une position de départ motrice D (θ_1 , θ_2) à une position d'arrivée A (θ_1' , θ_2') on va procéder de la méthode suivante :
 - On divise le déplacement entier du point D à A en petits déplacements.
 - On va choisir ce nombre de petits déplacements à chaque fois qu'on veut prédire la trajectoire.
 - Cette division du déplacement en coordonnées motrice nous permet d'avoir facilement une trajectoire Motrice
 - Après ça on va utiliser cette trajectoire motrice pour trouver une trajectoire spatiale en utilisant les fonctions d'estimation des coordonnées déjà développé dans la question 01 de cette partie avec les deux variantes.
- J'ai choisi cette solution en se basant sur le fait que les poids de nos neurones vont se rapprocher des données. Aussi, le fait que le mouvement du bras du robot est continu dans l'espace. Donc, grâce à notre carte et les fonctions de prédiction des coordonnées, on peut finalement estimer la trajectoire du bras du robot d'un point de départ à un point d'arrivée facilement.

- Vous trouverez dans code une fonction d'estimation de trajectoire nommée "Estimer Trajectoire Spatial" qui peut être paramétré avec un boolean passé en paramètres à la dernière position pour indiquer si on veut utiliser ou pas la moyenne pour passer d'un type de coordonné à un autre.
- Pour cette tâche de prédiction de trajectoire on trouve que la version de passage entre coordonnées avec la moyenne est plus performantes pour prédire une trajectoire vu qu' elle ne retourne jamais la même position spatiale pour deux différentes positions motrices contrairement à la version ou on n'utilise pas de moyenne.

Conclusion de la partie bras robotique:

Une fois j'ai fini d'implémenter les différentes fonction avec les différentes variantes j'ai voulu tout tester et calculer l'erreur moyenne de prédiction des coordonnées avec la version qui prend les coordonnées du neurones le plus proches et la deuxième qui prend en compte les quatres neurones voisins de notre entrée et fait une sorte de moyenne pondérée, vous trouverez les mesures dans les deux figures ci-dessous 9. (la carte a été entraîné avec les hypers paramètres suivantes ($\eta=0.005$, $\sigma= 1.4$, $N=30000$, **samples=1200**, **carte de taille (10*10)**) et les mesures de comparaisons vont dépendre sûrement de la qualité d'auto-organisation de la carte).

L'erreur moyenne calculée sur la figure 9 correspond à la moyenne de la distance entre l'estimation sortie par la méthode et le résultat attendu qu'on a pris à partir des données d'entrées.

Méthod -> Erreur moyenne de la méthode sur tout le jeu d'entrée sur plusieur exécutions	Estimations des coordonnées Motrice	Estimations avec moyenne des coordonnées Motrice	Estimations des coordonnées Spatial	Estimations avec moyenne des coordonnées Spatial
1	1.06	1.02	1.877	2.12
2	1.07	1.07	1.76	1.61
3	1.05	1.02	1.83	1.80
4	1.05	1.04	1.85	2.07

Figure 9 – Erreur moyenne sur les estimations des coordonnées pour les deux différentes méthodes.

On remarque dans le tableau 9 que les erreurs sur les deux différentes méthodes d'estimations sont très proches, mais on a tendance à dire que l'estimation d'une coordonnée spatiale en utilisant la moyenne des 4 neurones voisins à une erreur un peu plus élevée que la méthode où on prend juste le vecteur le plus proche.

On voit aussi que notre carte est meilleure peu importe la méthode utilisée pour estimer des coordonnées motrices.

Donc pour conclure, la méthode d'estimation de coordonnées en utilisant la moyenne nous apporte un avantage que pour l'estimation des trajectoires où on peut avoir plus de coordonnées différentes pour la trajectoire en utilisant la méthode avec la moyenne au lieu de la méthode avec le neurone le plus proche, mais elle reste moins précise que celle avec le neurone gagnant seulement.

PS:

Mon binôme SALMI DORIAN n'a pas pu continuer à travailler avec moi pour des bonnes raisons, depuis la première séance où on a fini la partie "**3.2/ Influence de σ** " ensemble.