

## TP Réseaux de neurones

### Partie 3- Etude "théorique" de cas simples

#### 3.1 Influence de $\eta$

*"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 ? Même question dans le cas où  $\eta = 1$ ."*

Par apprentissage, on entend l'évolution des poids d'un neurone entre les temps  $t$  et  $t+1$ .  
Si  $\eta = 0$ , alors l'apprentissage du neurone obtenu pour ce poids sera égale à 0, la prochaine valeur du poids du neurone gagnant restera donc la même.

Si  $\eta = 1$ , alors l'apprentissage du nouveau obtenu pour ce poids sera égale à  $e^{-\frac{\|j-j^*\|^2 c}{2\sigma^2}} (x_i - w_{ij})$   
 $\|j-j^*\|^2$  est une distance euclidienne entre le neurone auquel s'applique la fonction d'apprentissage et le neurone gagnant. Si notre neurone est le neurone gagnant en question, cette distance vaut 0. On a alors :  $e^{-\frac{\|j-j^*\|^2 c}{2\sigma^2}} = e^0 = 1$ , et  $\eta = 1$ , donc  $\Delta w_{i,j} = (x_i - w_{i,j})$   
Donc  $w_{i,j} = w_{i,j} + (x_i - w_{i,j}) = x_i$  Donc si  $\eta$  vaut 1, le nouveau poids du neurone gagnant sera égale au poids de la nouvelle entrée quelque soit son ancien poids.

*"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  $\eta$  (formule mathématique simple ou explication géométrique) ?"*

$e^{-\frac{\|j-j^*\|^2 c}{2\sigma^2}}$  étant une fonction de distance entre notre neurone et le neurone gagnant, et  $x_i - w_{ij}$  la différence entre le  $i$ ème élément du vecteur d'entrée et le  $i$ ème élément du vecteurs de poids de notre neurone.

En somme, l'apprentissage d'un poids d'un neurone correspondra à son écart à la valeur correspondante dans le vecteur d'entrée, multiplié par la distance entre son neurone et le neurone gagnant. A ce résultat, vas-être multiplié  $\eta$ , qui vas alors la réduire, et donc réduire la vitesse d'apprentissage globale du neurone, si elle se trouve entre 0 et 1.

Plus  $\eta$  est proche de 0, moins le neurone apprendra vite, plus  $\eta$  est proche de 1, plus le neurone apprendra vite.

### 3.2 Influence de $\sigma$

*“Si  $\sigma$  augmente, les neurones proches du neurone gagnant (dans la carte), vont-ils plus ou moins apprendre l’entrée courante ?”*

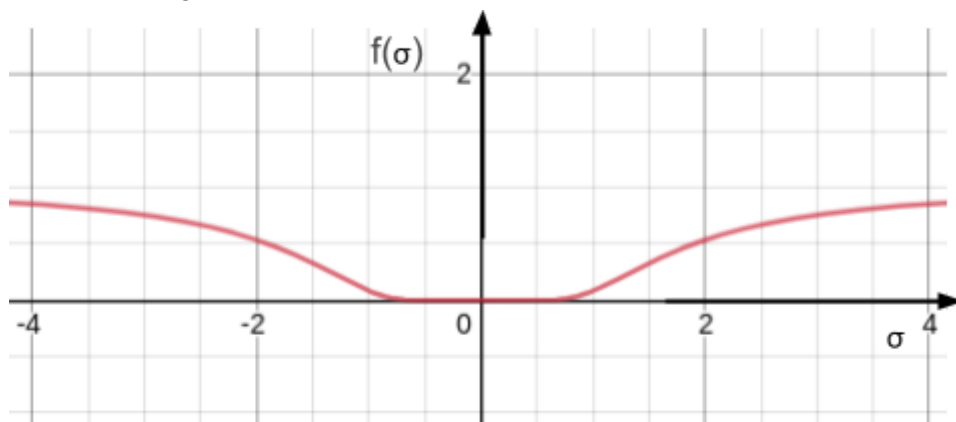
En sachant que  $\|j - j^*\|_c^2$  est une formule variable  $\geq 0$  car il s’agit d’une distance au carré.

Voici donc un tableau montrant l’analyse linéaire que nous pouvons faire de  $e^{-\frac{\|j-j^*\|_c^2}{2\sigma^2}}$  :

Valeurs de sigma	Tend vers $-\infty$	Tend vers 0 mais $\neq 0$	Tend vers $+\infty$
$\sigma^2$	Tend vers $+\infty$	Tend vers 0 mais $> 0$	Tend vers $+\infty$
Valeurs de $\frac{\ j-j^*\ _c^2}{2\sigma^2}$	Tend vers 0 mais $> 0$	Tend vers $+\infty$	Tend vers 0 mais $> 0$
Valeur de $-\frac{\ j-j^*\ _c^2}{2\sigma^2}$	Tend vers 0 mais $< 0$	Tend vers $-\infty$	Tend vers 0 mais $< 0$
Valeur de $e^{-\frac{\ j-j^*\ _c^2}{2\sigma^2}}$	Tend vers 1 mais $< 1$	Proche de 0 mais $> 0$	Tend vers 1 mais $< 1$

Nous avons considéré ici que  $\eta = 1$ , car c’est l’évolution de l’apprentissage en fonction de la distance au noeud gagnant qui nous intéresse ici, et non sa valeur. Faire varier  $\eta$  ne ferait que modifier la hauteur de la courbe, mais ne changerait pas sa forme.

Plus précisément, voici à quoi devrait ressembler la courbe de la fonction de distance en fonction de sigma :



Grâce à notre analyse représentée dans le tableau ci-dessus, nous pouvons remarquer que si  $\sigma$  augmente, les neurones proches du neurone gagnant vont plus apprendre l’entrée courante, puisque la fonction de distance vas se rapprocher de 1, limitant ainsi son impacte sur l’apprentissage.

De la même manière que précédemment, nous pouvons remarquer que plus la distance au neurone gagnant est grande, plus notre fonction de distance  $e^{-\frac{\|j-i\|^2}{2\sigma^2}}$  va être proche de 0, et donc moins le neurone en question va apprendre.

*“À convergence, si  $\sigma$  est (plus) grand, l’auto-organisation obtenue sera-t-elle donc plus “resserrée” ou plus “lâche” ?”*

Nous savons qu’une valeur de sigma plus grande fait tendre la fonction de distance vers 1 pour une même distance au neurone gagnant.

Prendre une valeur de sigma plus grande va alors réduire l’impact de la distance au neurone gagnant.

L’auto-organisation obtenue sera donc plus resserrée, car lorsqu’un neurone gagnant apprendra, ses voisins apprendront presque autant, et suivront plus facilement le neurone gagnant.

*“Quelle mesure pourriez-vous proposer pour quantifier ce phénomène et donc mesurer l’influence de  $\sigma$  sur le comportement de l’algorithme ?”*

Ce phénomène pourrait être observé à l’aide d’une fonction montrant la lâcheté du graphe.

Pour un graphe donné, il est possible de quantifier sa lâcheté  $L$  avec la formule :

$$L = \left[ \sum_{i=1}^n \left( \sum_{j=i}^n \frac{d(n_i, n_j)}{D(n_i, n_j)} \right) \right] / n!$$

Avec :

- $n$  le nombre de noeuds dans le graphe
- $n_i$  le noeud numéro  $i$  dans le graphe
- $d(n_i, n_j)$  la distance euclidienne dans le plan entre les noeuds  $n_i$  et  $n_j$
- $D(n_i, n_j)$  la distance euclidienne dans le graphe entre les noeuds  $n_i$  et  $n_j$

Nous cherchons donc à faire une moyenne des rapport entre les distances euclidienne dans le graphe et dans le plan pour chaque paires de neurones.

### 3.3 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  $\eta$  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) ?”*

Dans ce cas, le neurone étant le seul du graphe, il sera constamment neurone gagnant. Il apprendra donc avec une distance au neurone gagnant, la fonction de distance vaudra donc  $e^0 = 1$  :

$$\eta e^{-\frac{\|j-j^*\|^2 c}{2\sigma^2}} (x_i - w_{ij})$$

Ce qui nous donne

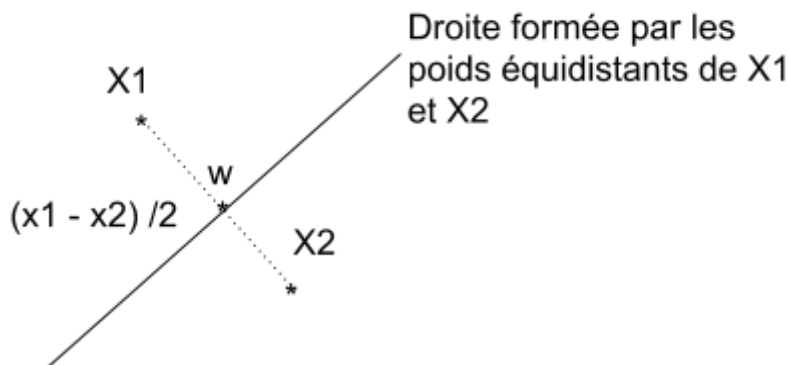
$$\|j-j^*\|^2 c = 0,$$

$$\Rightarrow \Delta w_{i,j} = \eta e^{-\frac{\|j-j^*\|^2 c}{2\sigma^2}} (x_i - w_{ij}) = \eta (x_i - w_{ij}).$$

Etant donné que  $\eta$  est considéré très petit, le neurone gagnant ne va que peu apprendre les poids des entrées. Il ne va donc pas se “téléporter” sur les entrées lors de leur apparition, et il faudra qu’une entrée apparaisse beaucoup de fois pour influencer son déplacement.

Sa position de départ influencera donc sa position au bout de  $m$  itérations, c’est à dire  $m$  apparitions de  $X1$  et  $X2$ . Cependant, à chaque itérations, le noeud se rapprochera de la paire  $X1, X2$ .

A partir du moment où notre neurone est suffisamment proche de ces deux entrées pour que leurs directions diverge, le neurone verra son poids se stabiliser sur le point le plus proche des poids des deux neurones, et ce sur la droite formée par les poids équidistants des deux neurones.



En effet, si le poids  $w$  de notre seul neurone est plus proche de l’un des deux points, alors il se déplacera moins dans la direction de ce dernier, et plus dans la direction de l’autre. En effet, plus notre poids  $w$  est loin de l’entrée, plus il s’en rapproche. Car dans ce cas,  $(x_i - w_{ij})$  est plus grand, et par conséquent,  $\Delta w_{i,j} = \eta (x_i - w_{ij})$  l’est aussi.

Le poids  $w$  se rapprochant des entrées à chacune de leurs apparitions, il se rapprochera du poids, de cette droite, le plus proche des entrées (droite en pointillés).

Enfin, nous pouvons dire que le poids  $w$  de notre neurone convergera vers le poids à mi-chemin entre  $X1$  et  $X2$ , c’est à dire du poids  $(x1 - x2) / 2$ .

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

Tout dépend de l'ordre de présentation de ces entrées.

1. Si X1 est présenté n fois, puis que les deux entrées sont présentées une infinités de fois, le poids  $w$  converge toujours vers le poids  $(x_1 - x_2) / 2$ .  
En effet, dans un premier temps, notre poids  $w$  se rapprochera beaucoup de X1 (nous pouvons imaginer que  $w$  et  $x_1$  sont collés).  
Cependant, nous avons vu précédemment que notre poids apprend plus un poids qui lui est éloigné qu'un poids qui lui est collé (évolution de  $\Delta w_{i,j} = \eta(x_i - w_{ij})$  en fonction de  $(x_i - w_{ij})$ ).  
Par conséquent, si on présente les entrées X1 et X2 à un poids très proche de X1, le poids en question se rapprochera plus de X2 que de X1. Le poids convergera donc vers  $(x_1 - x_2) / 2$ .
2. En revanche, si les entrées X1 et X2 sont présentées toutes deux m fois, puis-que X1 est présenté n fois seul, alors le poids  $w$  convergera vers X1, car il se rapprochera de X1 après avoir convergé vers  $(x_1 - x_2) / 2$ .

Pour conclure, si l'entrée X2 est présentée m fois, et que l'entrée X1 est présentée m + N fois, le poids  $w$  convergera vers un point entre X1 et  $(x_1 - x_2) / 2$ , ce point dépendant de l'ordre d'apparition des 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 ce phénomène.”*

Grâce aux observations faites à la question précédente, nous pouvons voir plus globalement que les neurones vont être plutôt proches des valeurs qui apparaissent le plus souvent, et non pas seulement se positionner en fonction des positions des entrées.  
Pour deux entrées, on doit donc considérer que les neurones se rapprocheront plutôt d'une entrée apparaissant plus souvent.

De cette manière, si on imagine une zone A, très dense en données, et une zone B moins dense, il y aura plus de chance de trouver des données provenant de la zone A que des données provenant de la zone B parmi les données tirées aléatoirement.  
Par conséquent, les neurones se dirigeront majoritairement vers la zone A, plus dense.

## Partie 4 - Étude pratique

### 4.2 Implémentation

Afin de pouvoir être intégré dans une boucle pour l'évaluation des performances, le code lançant la simulation et l'apprentissage a été placé dans des fonctions dans le fichier "src/simulation.py".

Pour lancer une simulation, lancer la commande ``python3 src/run.py``.

Voici la liste des options à ajouter à cette commande pour obtenir la simulation désirée :

- ``-h`` (hidden) désactive l'option verbose, supprime l'affichage pour une simulation plus rapide.
- ``-r`` pour lancer un jeu de données du bras robotique.
- ``-d <num>`` lance la simulation sur le jeu de données numéro <num>. Si ce jeu de données n'est pas connu, utilise le jeu de données 1 par défaut.
- ``-rect x y`` Pour préciser que le graph de neurones est un rectangle de taille x y. Par défaut, le graph de neurones est un rectangle de taille 10 par 10.

### 4.3 Analyse de l'algorithme

*Pour réaliser l'étude pratique du comportement de l'algorithme (à mettre en regard de votre étude théorique faite dans la section 3), étudiez 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 plus la mesure d'auto-organisation que vous avez proposé à la section 3.2)*

Pour chacun des éléments à étudier, nous allons comparer leur différentes valeurs de test. Pour chaque exécution, seul le paramètre étudié verra sa valeur remplacée dans une simulation dont les autres paramètres sont initialisés avec une valeur par défaut. Il aurait été plus pertinent de tester toutes les combinaisons possibles parmi les valeurs à tester pour chaque paramètre, mais cela imposerait un trop grand nombre de simulation.

Voici les paramètres par défaut que nous allons utiliser :

```
GRAPH_DIMENSIONS = (10, 10)
ETA = 0.1          # Taux d'apprentissage
SIGMA = 1.4        # Largeur du voisinage
N = 30000          # Nombre de pas de temps d'apprentissage
DATASET_SIZE = 999 # Taille du jeu de données
```

N.B. : L'attribut N sera évalué tout au long de chaque simulation.

Nous découperons les simulations en plusieurs phases égales. Ainsi, la taille maximale de N étant fixé à 40 000, le programme de simulation ne retournera pas un seul résultat, mais une liste de plusieurs résultats, nous permettant non seulement d'étudier la vitesse de l'apprentissage, mais également de trouver une valeur optimale pour N.

Chaque paramètres sera évalué sur plusieurs jeux de données :

1. Le jeu de données n°1 de l'énoncé (Carré)
2. Le jeu de données du bras robotique
3. Un jeu de données formant un cercle

4. Un jeux de données formant une ligne
5. Un jeux de données formant un triangle

Tous les résultats ne seront pas présentés ici, mais les données obtenues sont disponible ici :

[https://docs.google.com/spreadsheets/d/1OXJgpJGL\\_rvd-0w1iNWzVmHlp3SPuzUfBoE1XZ-mdTg/edit?usp=sharing](https://docs.google.com/spreadsheets/d/1OXJgpJGL_rvd-0w1iNWzVmHlp3SPuzUfBoE1XZ-mdTg/edit?usp=sharing)

Ou dans l'archive finale sous forme d'un tableur.

Taux d'apprentissage  $\eta$

Evolution de l'erreur de quantification vectorielle moyenne en fonction de la durée de l'apprentissage pour différentes valeurs de  $\eta$

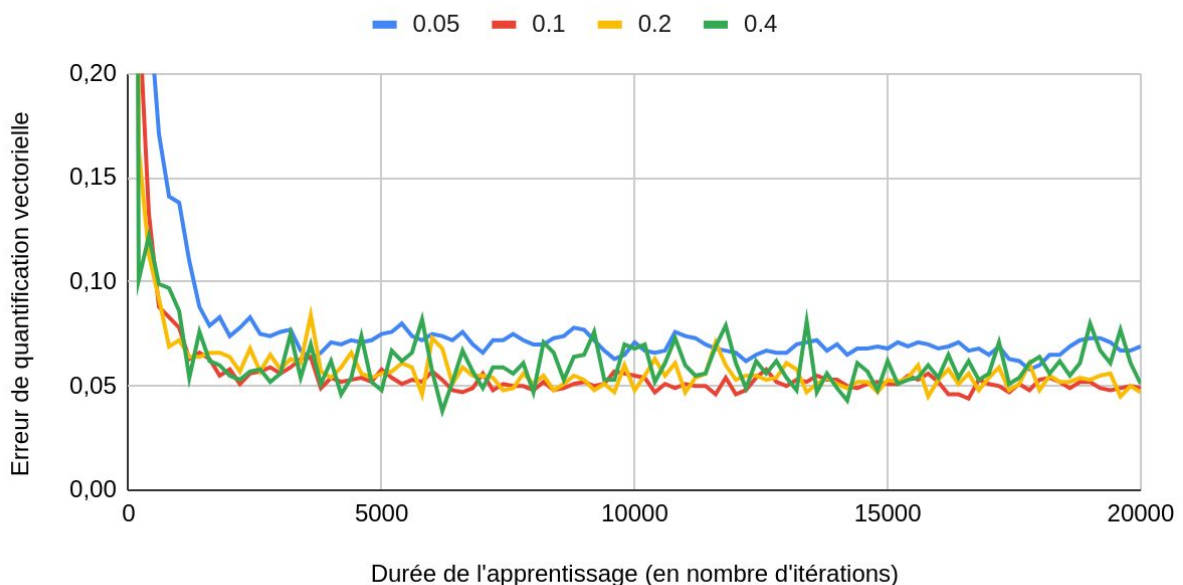


figure 1 : Simulations réalisés sur le dataset du bras robotique

Ce graphique (figure 1) représente l'évolution de l'erreur de quantification vectorielle moyenne (EQVM) au fil de l'apprentissage pour les valeurs de test de  $\eta$ , soit [0,05; 0,1; 0,2; 0,4].

La hauteur du graphique a été tronquée à 0,2 car l'EQVM du graphe initialisé aléatoirement n'est pas pertinent.

De ce graphique, nous pouvons remarquer deux conséquences majeurs de l'évolution de  $\eta$ . Tout d'abord, nous pouvons remarquer qu'augmenter la valeur de  $\eta$  ne fait pas qu'améliorer la vitesse de l'apprentissage, mais améliore également son erreur moyenne.

Valeur	0.05	0.1	0.2	0.4
Valeur moyenne	0,1033	0,0815	0,0848	0,0753
Valeur finale	0,069	0,049	0,047	0,051

figure 2

Chaque simulation comprend 100 évaluation du graph, à la fois de l'erreur de quantification vectorielle moyenne, mais également de la mesure d'auto-organisation. Les valeurs moyenne et finales présentés dans la figure 2 correspondent aux valeurs moyennes et finales pour ces cent évaluations pour chacune des quatres simulations présentés dans le graphe en figure 1.

Ces valeurs nous montre qu'à partir de  $\eta = 0.1$ , l'apprentissage ne gagne pas significativement en efficacité. De plus le graph (figure 1) nous permet de constater que plus  $\eta$  est grand, plus l'EQVM perd en régularité. Pour souligner ce fait, nous pouvons remarquer que si la simulation avait-été arrêtée 400 itérations plus tôt (1 mesure toutes les 200 itérations), l'EQVM mesurée pour  $\eta = 0,4$  aurait été très grande.

N	0.05	0.1	0.2	0.4
19400	0,071	0,048	0,056	0,061
19600	0,067	0,049	0,045	0,077
19800	0,067	0,05	0,05	0,061
20000	0,069	0,049	0,047	0,051

*figure 3 : Évolution de l'EQVM sur la fin des simulations*

Nous pouvons donc dire qu'au niveau de précisions que nous avons ici pour les valeurs de  $\eta$ , la valeur optimale pour ce paramètre est 0,1. En effet, cette valeur est celle pour laquelle l'EQVM est le plus faible parmi les mesures les plus régulières.

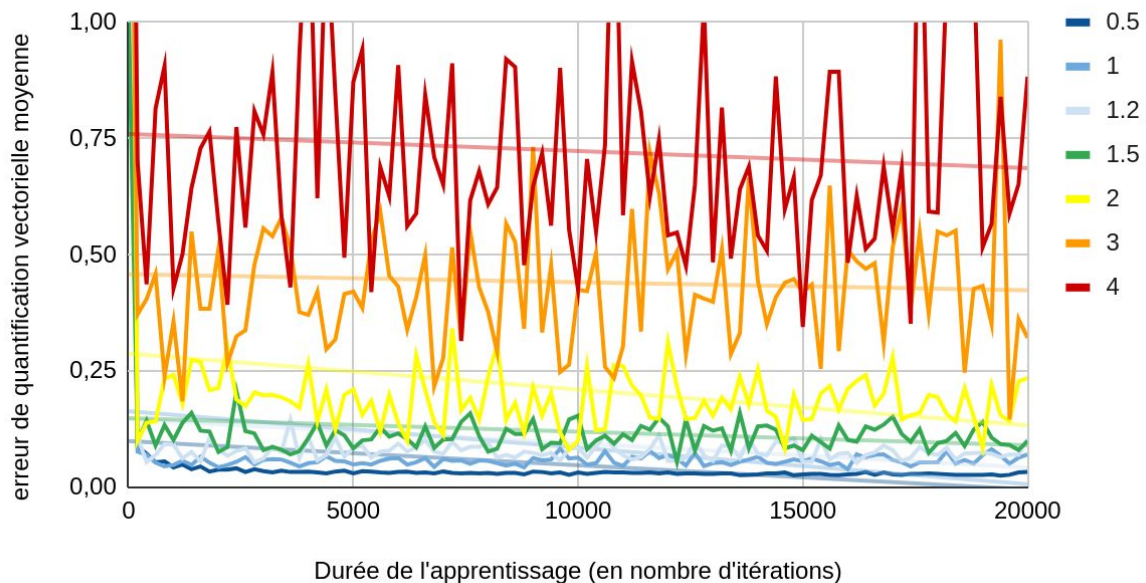
Afin d'approfondir nos résultats, nous pourrions ajouter des simulations pour des valeurs plus précises entre 0,1 et 0,2, ou encore établir une formule mesurant le niveau de régularité de l'apprentissage (ecart-type) afin d'en comparer la valeur pour chaque simulations.



Largeur du voisinage  $\sigma$

*Etude de l'impacte de sigma sur l'erreur de quantification vectorielle moyenne (EQVM)*

Evolution de l'erreur de quantification vectorielle moyenne en fonction de la durée de l'apprentissage pour différentes valeurs de sigma



*figure 4 : graphique montrant l'évolution de l'EQVM en fonction de sigma*

Sur ce graphique (figure 4), nous pouvons observer l'impacte de la valeur de sigma l'EQVM. Nous pouvons observer que plus sigma est élevé, plus l'EQVM est instable. En effet, si sigma est élevé, la largeur du voisinage apprenant à chaque itération sera plus grand, et ainsi, un grand nombre de neurones vont se diriger vers cette nouvelle entrée, déplaçant ainsi une plus importante partie du graphe, en empêchant une répartition équitable du graphe sur les entrées.

Ce phénomène rend également l'apprentissage inefficace, ce que l'on peut observer sur notre graphe. Nous pouvons y remarquer que plus la valeur de sigma est basse, plus l'EQVM.

Evolution de l'erreur de quantification vectorielle moyenne en fonction de la durée de l'apprentissage pour différentes valeurs de sigma

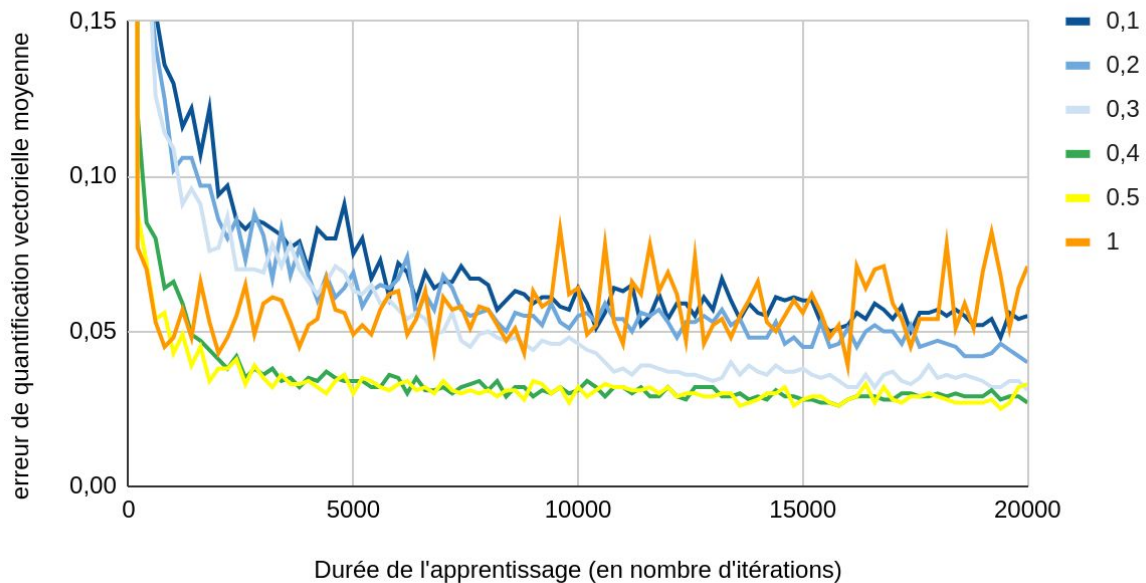


figure 5 : version plus précise de la figure 4

Le graphe de la figure 4, en plus d'être peu précis pour les valeurs de sigma faible, ne nous permettait pas de voir à partir de quel valeur de sigma l'apprentissage devenait inefficace.

**Attention : sur cette nouvelle version, la légende est très différente.**

Avec cette nouvelle version, nous pouvons constater que l'apprentissage n'est plus optimale pour des valeurs de sigma en dehors de  $]0,4; 0,5[$ .

Cependant, la modification de sigma a d'autres impacts sur notre apprentissage.

*Etude de l'impacte de sigma sur la lâcheté du graphe*

Pour mesurer l'impacte de sigma sur la lâcheté du graphe, nous avons utilisé la mesure d'auto-organisation présenté à la question 3.2.

Evolution de la mesure d'auto-organisation du graphe en fonction de la durée d'apprentissage pour différentes valeurs de sigma

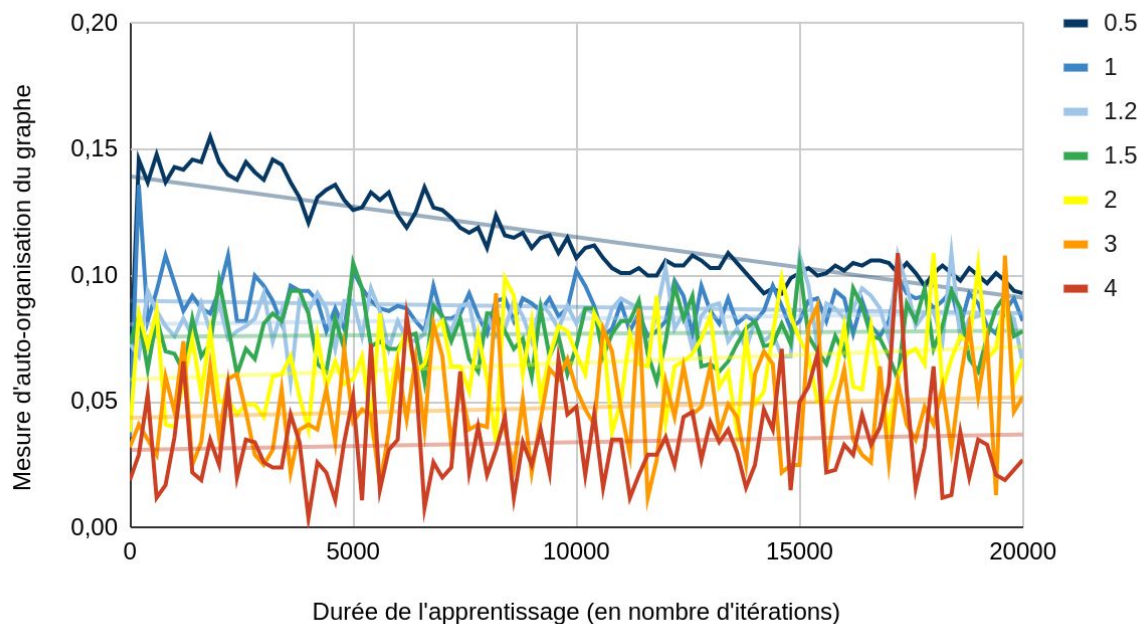


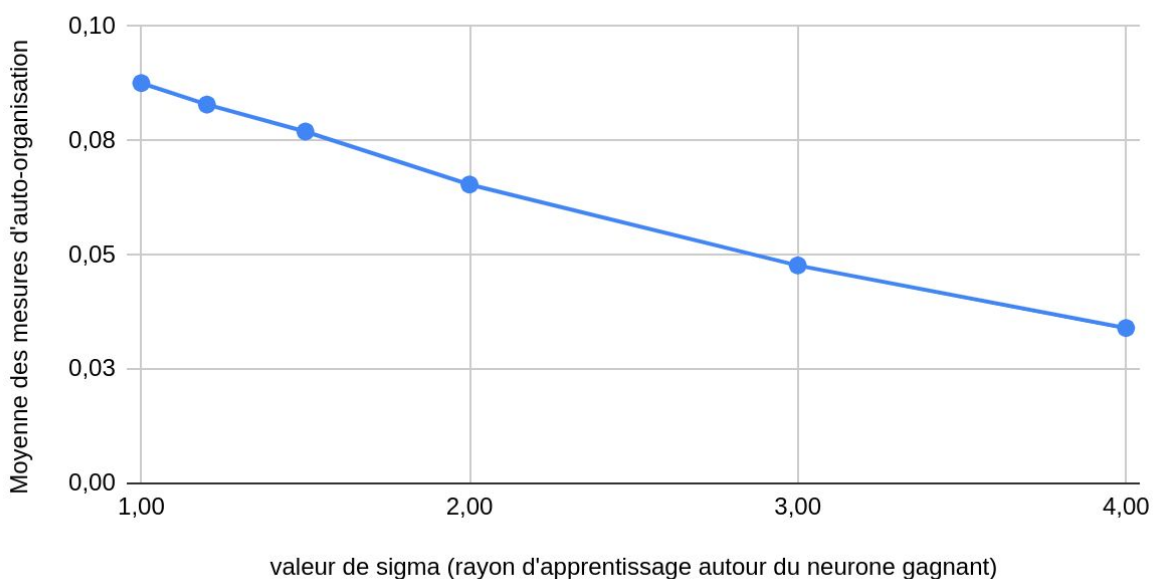
figure 6 : évolution de l'auto-organisation du graphe en fonction de sigma

Ce dernier graphe (figure 6) montre l'évolution de la mesure d'auto-organisation en fonction de sigma.

Il est important de noter que plus cette mesure est élevée, plus le graphe est lâche.

Ce graphique est intéressant car plus précis, mais les valeurs d'auto-organisations en fonction de sigma seront suffisante pour tirer des conclusions.

Evolution de la mesure d'auto-organisation moyenne en fonction de sigma

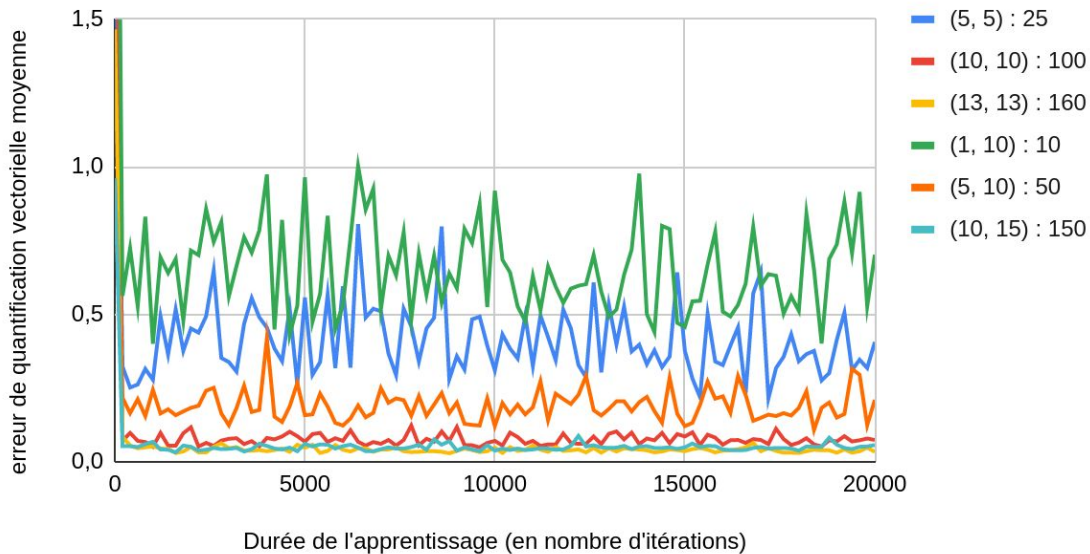


*figure 7 : Auto-organisation moyenne en fonction de sigma*

Ce dernier graphique (figure 7) nous permet de confirmer nos conclusions tirées à la question 3.2, à savoir que plus sigma est élevé, plus le graphe est resserré.

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

Evolution de l'erreur de quantification vectorielle moyenne en fonction de la durée d'apprentissage pour différentes dimensions de graphe.



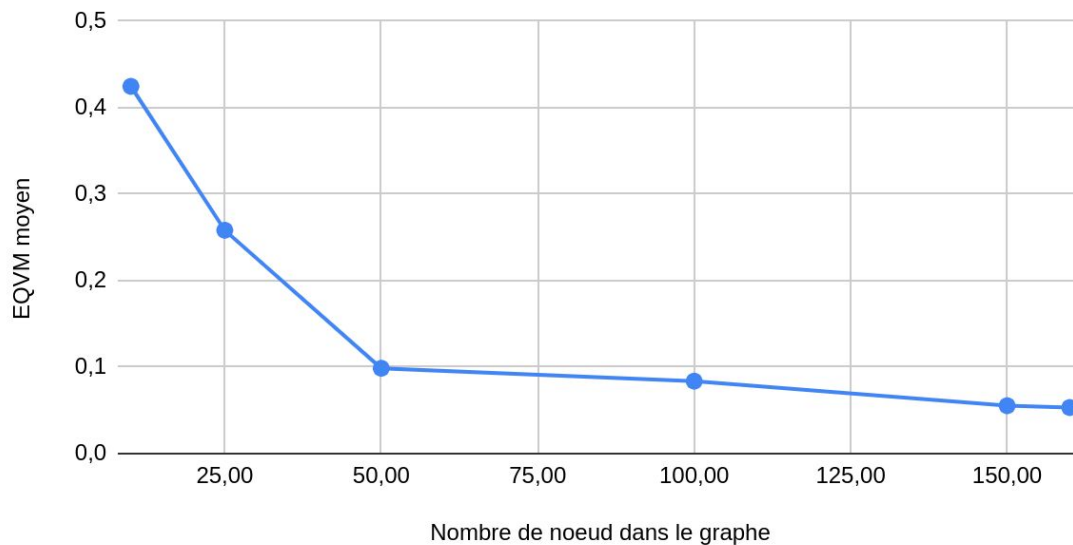
*figure 8 : évolution de l'EQVM pour différentes dimensions*

Sur ce graphe, nous pouvons remarquer différentes mesures de l'EQVM sur des simulations avec différentes dimensions de graphe.

Ce graphe ne nous montrent que nos simulations ne nous montrent aucun impacte notable de la forme du graphe (carré/rectangle/ligne) sur le dataset du robot qui ne semble pas avoir de forme remarquable. En effet, le petit carré (bleu) est moins efficace que le grand rectangle (cyan) alors que le grand carré (jaune) est plus efficace que le petit rectangle (orange).

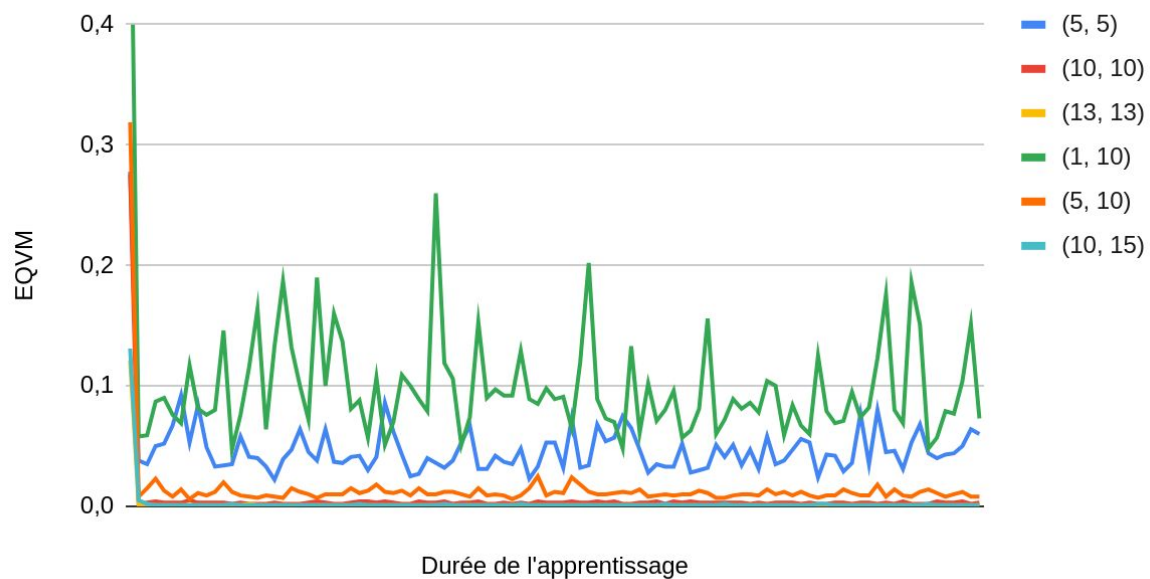
Cependant, ce graphique semble souligner une différence en terme de nombre de neurones dans le graphe (indiqué dans la légende en dehors des parenthèses).

### Erreur de quantification vectorielle moyenne par rapport au nombre de noeud dans le graphe



Grâce à ce graphique, nous pouvons remarquer que plus il y a de neurones dans le graphe, plus leur répartition par rapport à l'ensemble d'entrée sera précis, et plus l'EQVM sera faible.

### Evolution de l'EQVM en fonction des dimensions du graphe sur un ensemble de donnée en forme de cercle



Ce graphique nous permet de remarquer que même sur un jeu de données dans lequel la structure en ligne ne devrait pas-être désavantagé (car l'ensemble des données d'entrée forme une ligne et non une surface), son erreur de quantification vectorielle moyenne est très grande.

Cela est dû au fait que les structures rectangulaires s'adaptent très bien aux lignes.

En effet, on peut transformer un rectangle en ligne en utilisant la même valeur pour tous les poids en x (ou tous les poids en y).

Concernant l'ensemble de données en triangle, la moyenne des valeurs de l'EQVM pour chacune des formes de graphe est la même (à  $\pm 10^{-4}$  près) que pour notre ensemble de données en cercle.

Le nombre de neurones dans le graphe a donc plus d'importance que la corrélation entre la forme du graphe et la forme de l'ensemble de données.

Pour finir notre analyse, nous pouvons ajouter qu'il est inutile d'utiliser une valeur de N supérieure à 5000 (10 000 si la valeur de sigma est inférieure à 0,4) tant que les valeurs des paramètres sont équivalentes à celles que nous avons observé.

En effet, au-delà de cette valeur de N, l'apprentissage se stabilise et le réseau de neurones ne converge plus (les seules évolutions observables après ce cap, sont aléatoires et irrégulières).

Pour aller plus loin, nous pourrions imaginer des algorithmes faisant évoluer ces paramètres au cours de l'apprentissage, comme une valeur de sigma plus grande au début de l'apprentissage pour donner la forme globale du réseau puis de plus en plus faible pour avoir un apprentissage plus précis, ou encore un algorithme qui ajoute des nœuds et des liaisons s'il détecte que des neurones apprennent plus souvent que d'autres, afin de combler les zones n'étant pas couvertes par suffisamment de neurones, dans le cas où la structure du réseau ne coïncide pas avec la forme de l'ensemble d'entrée.

#### 4.4 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 ?*

Dans les deux cas, pour prédire un sous-ensemble de données d'apprentissage étant donné un autre, il suffit de trouver le neurone gagnant selon les données que l'on connaît, et de récupérer celles qu'on ne connaît pas sur ce neurone gagnant.

Par exemple, si on connaît les angles et que l'on cherche la position de la main, nous allons commencer par chercher le neurone ayant des poids pour les angles le plus proches des angles que nous connaissons, et nous trouverons la position de la main avec les poids  $x_1$  et  $x_2$  de ce neurone, et inversement si l'on cherche l'angle à partir de la position de la main.

A partir de cette méthode, nous pouvons ajouter des méthodes permettant de contraindre l'erreur obtenue entre le neurone gagnant et les données connues.

Par exemple, on peut imaginer prendre un ensemble de plusieurs neurones gagnants (les trois meilleurs par exemple) attribuer un certain niveau de réussite à chacun (100 - le pourcentage de leur taux d'erreur dans la somme des trois taux d'erreur par exemple), et



utiliser ces taux de réussite comme poids d'une moyenne pondérée des données que nous recherchons sur celles des neurones gagnant.

**Edit :** Le problème de ce calcul, est que l'importance de chacun des meilleurs neurones est calculée en fonction de leur erreur par rapport à l'erreur globale. Ainsi, si un neurone n'a fait aucune erreur ou une erreur très faible (0 ou 0,000001), qu'un second a une erreur faible (0,1) et que le troisième a une erreur très grande (par exemple 4), les deux premiers neurones auront une importance quasiment équivalente à cause du troisième alors que le second neurone aura un effet négatif sur le premier (Si un neurone a une erreur nulle, on ne veut pas prendre en compte les autres quel que soit leur erreur).

Une meilleure méthode pour calculer leur importance, serait alors d'observer leur rapport d'erreur deux à deux, et d'en calculer le produit. Ainsi, si un des neurones a une erreur très faible, l'importance des autres sera aussi très faible.

$$\forall n \in G, \text{imp}(n) = \prod_{i=1}^m \frac{E_{n_i}}{E_n} \quad \text{avec :}$$

- $G$  notre ensemble de neurones gagnants,
- $\text{imp}(n)$  l'importance du neurone  $n$ ,
- $E_n$  l'erreur du neurone  $n$ ,
- $\{n_1, \dots, n_m\} = G \setminus \{n\}$ ,

*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 ?*

Le modèle de Hopfield pourrait nous permettre de restituer une partie d'une entrée à partir d'une autre.

Cette méthode serait plus fidèle à l'ensemble d'entrées, car les neurones ne vont pas se positionner de manière à se trouver au plus près des entrées, mais vont les apprendre "par cœur". L'erreur (écart entre la prédiction et la réalité) sera donc plus faible.

En revanche, ce modèle ne peut apprendre qu'un nombre d'entrées très limités (jusqu'à 0,14 \* le nombre de neurones). Pour apprendre les 999 entrées de nos ensembles de tests, nous aurions donc eu besoin de 7136 neurones.

*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 ?*

Un perceptron (multi-couches ou non) pourrait nous permettre de prédire une position spatiale à partir de deux angles.

En revanche, son utilisation ne sera pas à double sens comme c'était le cas avec les autres modèles.

De plus, ce modèle présente l'inconvénient de ne pas offrir de garantie de convergence dans son apprentissage, ni de garantie quant à son efficacité.

Il présente cependant l'avantage de pouvoir s'adapter à n'importe quel structure d'entrées (contrairement au modèle de kohonen qui est plus efficace avec un réseau de neurone qui présente la même structure que l'ensemble de données d'entrées).

*On veut déplacer le bras d'une position motrice  $(\theta_1, \theta_2)$  à une nouvelle  $(\theta'_1, \theta'_2)$ . 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.*

Comme expliqué à la première question, il est possible d'obtenir une position spatiale de la main à partir d'une position motrice.

Pour obtenir une trajectoire des positions spatiales de la main, à partir d'une évolution des positions motrices, une méthode pourrait-être de découper l'évolution de la position motrice en étapes d'intervalles égales pour obtenir une liste de positions spatiales de la main.

En reliant les positions spatiales obtenues, il est donc possible d'obtenir un tracé du mouvement de la main.

*(Bonus) Modifiez/Complétez le code fourni pour apprendre sur les données robotique et implémentez les mécanismes de prédiction (d'une position motrice (resp. spatiale) à partir d'une configuration spatiale (resp. motrice) et de la trajectoire de la position de la main lors d'un mouvement).*

Pour demander au code de trouver des positions spatiales à partir d'angles pour le bras robotique, ou inversement trouver des angles à partir d'une position spatiale, exécutez la commande :

```
python3 src/test_robot_arm.py
```

dans le fichier test\_robot\_arm.py, se trouve deux listes de test. Le programme vas créer et entraîner un réseau de neurone, puis lui demander de compléter les données présentes dans les listes de test.

A chaque test, le programme affiche le test de départ, le résultat renvoyé par le réseau de neurone, et l'erreur commise, c'est à dire, dans l'ensemble de quatres poids reconstitué par le réseau, la distance entre le point donné par les angles et le point donné par le réseau de neurone.

Cette commande lance également les tests pour la prédiction de trajectoire.

## Sources

Représentation graphiques en ligne des fonctions :

<https://www.desmos.com/calculator?lang=fr>