Intelligence Artificielle
Réseaux de neurones

Mathieu Lefort

23, 25 mars 2022

Ce projet peut être fait en binôme. Vous rendrez à la fin de chaque séance votre état d'avancement. La version finale est à déposer pour le 6 mai.

1 Objectif

L'objectif est de comprendre et d'implémenter l'algorithme de carte auto-organisatrice de Kohonen.

2 Algorithme

Pour rappel l'algorithme de Kohonen est le suivant :

- Initialisation
  1. Initialisation des poids à des valeurs aléatoires
  2. Choix de la métrique dans l'espace d'entrée (par défaut on prendra une distance euclidienne $||\cdot||$)
  3. Choix de la taille et de topologie de la carte (par défaut on prendra une grille carrée avec une distance euclidienne $||\cdot||_c$ et un voisinage gaussien)
  4. Choix des hyperparamètres (largeur du voisinage gaussien σ, taux d'apprentissage η et nombre de pas d'apprentissage N)
- Apprentissage
  Pendant N pas de temps
  1. Choix d'une nouvelle entrée $\mathbf{X}$
  2. Pour chaque neurone j, calcul de la distance $||\mathbf{W_j} - \mathbf{X}||$ entre son poids $\mathbf{W_j}$ et $\mathbf{X}$
  3. Détermination du neurone gagnant j^*, i.e. le plus proche de l'entrée courante ($||\mathbf{W_j^*} - \mathbf{X}|| = \min_j ||\mathbf{W_j} - \mathbf{X}||$)
  4. Pour chaque neurone j, mise à jour des poids de toutes ses connexions [1] : $\Delta w_{ij} = \eta e^{-\frac{||j-j^*||_c^2}{2\sigma^2}}(x_i - w_{ij})$

3 Étude "théorique" de cas simples (5.5 points)

L'idée est ici de bien comprendre les règles d'apprentissage, afin de "prédire" l'influence des différents paramètres, que vous testerez en pratique dans la section 4. Pour cette partie, chaque réponse devra être justifiée, même si une démonstration mathématique formelle n'est pas indispensable.

3.1 Influence de η

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

- Dans le cas où $\eta = 0$ quelle sera la prochaine valeur des poids du neurone gagnant ?
- Même question dans le cas où $\eta = 1$.
- Dans le cas où $\eta \in]0, 1[$ (paramétrisation "normale") où se situera le nouveau poids par rapport à $\mathbf{W}^*$ et $\mathbf{X}$ en fonction de η (formule mathématique simple ou explication géométrique) ?
- Que va-t-il se passer si $\eta > 1$?

1. Puisque la carte de neurone est carrée, si un neurone j est à la position (x_j, y_j) dans la carte, alors $||j - j^*||_c = ||(x_j, y_j) - (x_{j^*}, y_{j^*})||_c$.

3.2 Influence de σ

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

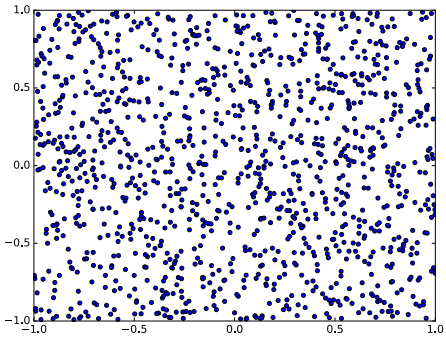
3.3 Influence de la distribution d'entrée

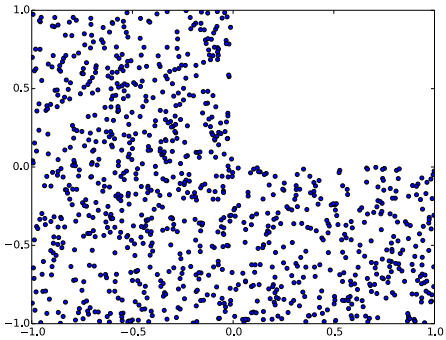
— Prenons le cas très simple d'une carte à 1 neurone qui reçoit deux entrées $\mathbf{X_1}$ et $\mathbf{X_2}$.
  — Si $\mathbf{X_1}$ et $\mathbf{X_2}$ sont présentés autant de fois, vers quelle valeur convergera le vecteur du poids du neurone (en supposant un η faible - pour ne pas avoir à tenir compte de l'ordre de présentation des entrées - et suffisamment de présentations - pour négliger l'influence de l'initialisation des poids) ?
  — Même question si $\mathbf{X_1}$ est présenté n fois plus que $\mathbf{X_2}$.
— 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.

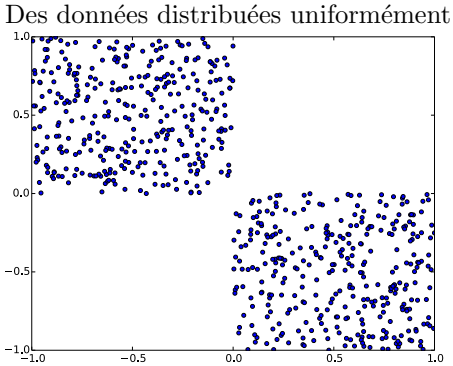
4 Étude pratique

4.1 Données

Vous avez à disposition plusieurs jeux de données :

1. Des données distribuées uniformément dans le quadrant $[-1, 1] \times [-1, 1]$:


2. Des données distribuées uniformément dans les quadrants $[-1, 0] \times [0, 1]$, $[-1, 0] \times [-1, 0]$ et $[0, 1] \times [-1, 0]$:


3. Des données distribuées uniformément dans les quadrants $[-1, 0] \times [0, 1]$ et $[0, 1] \times [-1, 0]$:


4. Des données correspondant à un bras robotique simplifié et qui seront détaillées dans la section 4.4

4.2 Implémentation (2 points)

Un fichier *kohonen.py* vous est fourni et contient une partie affichage et le squelette de l'algorithme (il ne manque que les équations de calcul d'activité et d'apprentissage des poids). Vous pouvez lancer le fichier avec la commande suivante : *python kohonen.py*[2].

Ce code propose les 2 types d'affichages suivants :

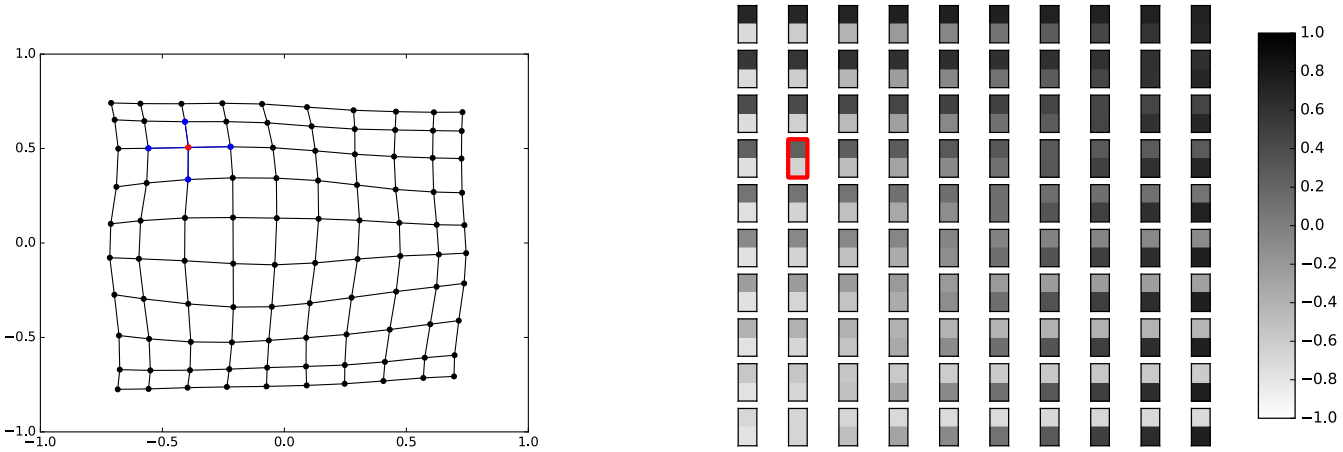


FIGURE 1 – Affichages obtenus après apprentissage sur le jeu de donnée 1

Sur la figure de gauche (dans l'espace d'entrée), le poids de chaque neurone est représenté par un point et ses voisins (à distance 1) dans la carte sont représentés par un lien entre leurs poids respectifs (représentation vue en CM). Par exemple, le neurone à la position (3,3) dans la carte a les poids représentés par le cercle rouge (ici [-0.4,0.5]) et les neurones aux positions (4,3), (2,3), (3,4) et (3,2) ont les poids indiqués par les cercles bleus.

Sur la figure de droite (dans l'espace de la carte) sont affichés les poids de chaque neurone. Par exemple, le rectangle rouge indique les poids du neurone à la position (4,2) dans la carte, la valeur des poids étant indiquée en niveau de gris (ici environ [0.4,-0.6]))

Le programme affiche également l'erreur de quantification vectorielle faite par le réseau sur le jeu d'apprentissage $\{\mathbf{X}\} : \frac{1}{|\{\mathbf{X}\}|} \sum_{\mathbf{X} \in \{\mathbf{X}\}} \min_j ||\mathbf{W_j} - \mathbf{X}||^2$.

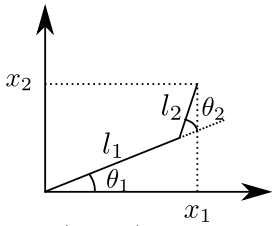
— Implémentez l'algorithme (les sections TODO vous indiquent les sections de code à modifier/écrire). Vous privilégierez au maximum l'utilisation de la librairie de calcul matriciel numpy qui fournit des performances d'exécutions bien supérieures à l'utilisation des boucles *for*. Vous pouvez lancer le fichier tuto_numpy.py pour avoir une liste de commande numpy utiles et/ou regarder la documentation en ligne `http://docs.scipy.org/doc/numpy-1.10.1/reference/index.html`. Le paramétrage qui vous est fourni dans le code devrait fonctionner si vous respectez les équations demandées.

4.3 Analyse de l'algorithme (5 points + 2 points bonus)

— 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 et avec la mesure d'auto-organisation que vous avez proposé à la section 3.2) :
  — taux d'apprentissage η
  — largeur du voisinage σ
  — nombre de pas de temps d'apprentissage N
  — taille et forme de la carte (vous pouvez tester facilement des formes 'lignes', 'carrées' et 'rectangles')
  — jeu de données. En particulier créez vos propres jeux de données avec des données non uniformément distribuées pour étudier la répartition des poids des neurones.
  — (Bonus) la topologie de la carte. Par exemple, au lieu d'utiliser une grille carrée, utilisez une grille hexagonale.

2. Pour le lancer il vous faut préalablement installer python et les librairies numpy et matplotlib.

4.4 Bras robotique (7.5 points)



Nous considérons un bras robotique composé d'un bras de longueur l_1 et d'un avant bras de longueur l_2 se déplaçant dans le plan. En pratique vous n'auriez pas (forcément) accès à ces informations, uniquement aux données enregistrées.

On veut apprendre la relation entre la commande motrice (θ_1, θ_2) et la position de la main (x_1, x_2). Pour cela on va apprendre une carte de Kohonen prenant en entrée des exemples de quadruplet $(\theta_1, \theta_2, x_1, x_2)$, qui pourraient être générés en déplaçant physiquement le bras et en observant sa position. Dans les données ci-dessous toutes les configurations motrices sont réalisables, mais ce n'est pas forcément toujours le cas.

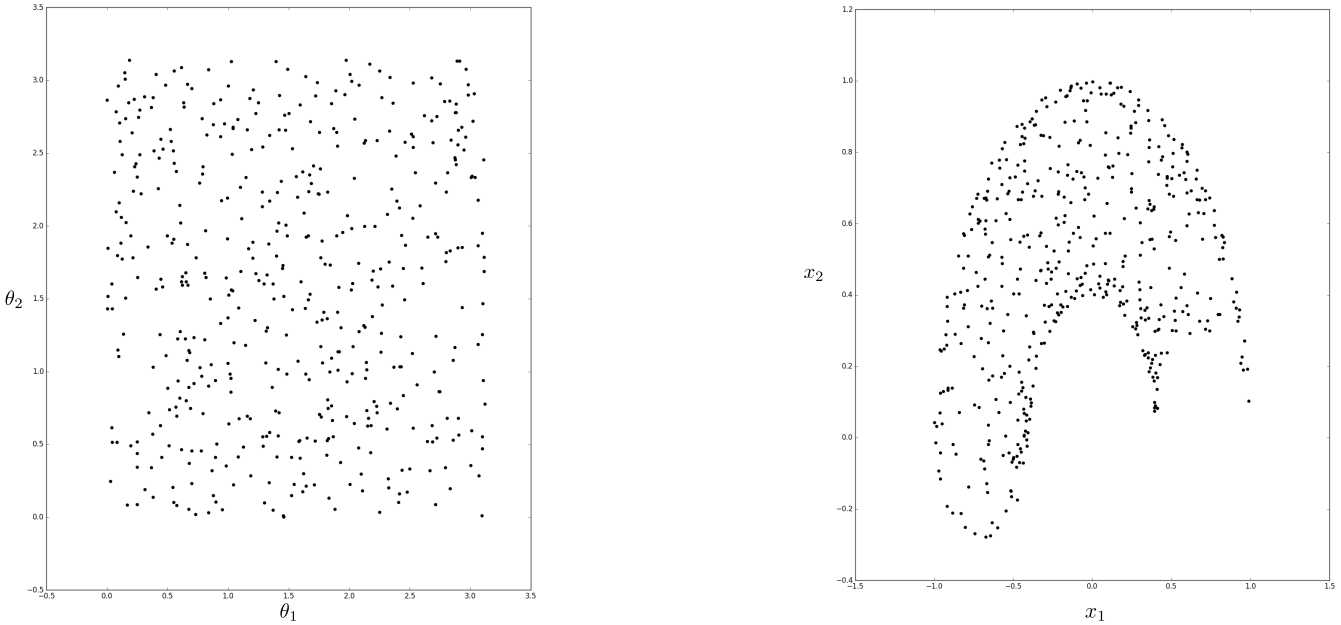


FIGURE 2 – Exemples de données de la position motrice du bras (à gauche) et de la position de la main dans l'espace (à droite), dans le cas où $l_1 = 0.7$ et $l_2 = 0.3$

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