

# Techniques d'Intelligence Artificielle

## Réseaux de neurones

Mathieu Lefort

28 mai 2020

Ce projet est à faire de manière individuelle (cela n'empêche pas de vous entraider, mais le rendu et donc la compréhension doivent être personnels). Vous rendrez à la fin de chaque séance votre état d'avancement. La version finale est à déposer pour le 21/06/20.

## 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 ( $\|\cdot\|$ )
  3. Choix de la taille et de topologie de la carte (ici on prendra une grille carrée avec une distance euclidienne  $\|\cdot\|_c$  et un voisinage gaussien)
  4. Choix des paramètres (largeur du voisinage gaussien  $\sigma$ , taux d'apprentissage  $\eta$  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<sup>1</sup> :  $\Delta w_{ij} = \eta e^{-\frac{\|j-j^*\|_c^2}{2\sigma^2}} (x_i - w_{ij})$

## 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 $\eta$

On notera l'entrée courante  $\mathbf{X}$  et 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  $\eta$  (formule mathématique simple ou explication géométrique) ?

---

1. Puisque la carte de neurone est carrée, si un neurone  $j$  est à la position  $(x_j, y_j)$  dans la carte, alors  $\|j - j^*\|_c = \|(x_j, y_j) - (x_{j^*}, y_{j^*})\|_c$  (ici égale à  $\sqrt{(x_j - x_{j^*})^2 + (y_j - y_{j^*})^2}$  puisque la distance dans la carte est euclidienne).

### 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 ?
- À convergence, si  $\sigma$  est (plus) grand, l'auto-organisation obtenue sera-t-elle donc plus “resserrée” ou plus “lâche” ?
- Quelle mesure pourriez-vous proposer pour quantifier ce phénomène et donc mesurer l'influence de  $\sigma$  sur le comportement de l'algorithme ?

### 3.3 Influence de la distribution d'entrée

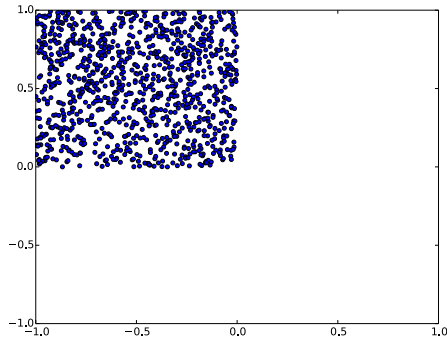
- Prenons le cas très simple d'une carte à 1 neurone qui reçoit deux entrées  $\mathbf{X}_1$  et  $\mathbf{X}_2$ .
  - Si  $\mathbf{X}_1$  et  $\mathbf{X}_2$  sont présentés autant de fois, vers quelle valeur convergera le vecteur du poids du neurone (en supposant un  $\eta$  faible pour ne pas avoir à tenir compte de l'ordre de présentation des entrées et suffisamment de présentations pour négliger l'influence de l'initialisation des poids) ?
  - 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 ce phénomène.

## 4 Étude pratique

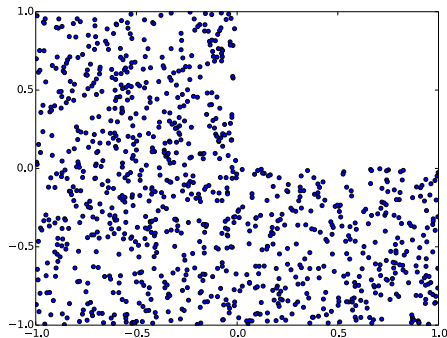
### 4.1 Données

Vous avez à disposition plusieurs jeux de données :

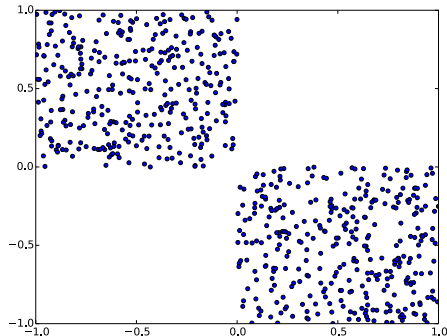
1. Des données distribuées uniformément dans le quadrant  $[-1, 0] \times [0, 1]$  :



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`<sup>2</sup>.

Ce code propose les 2 types d'affichages suivants :

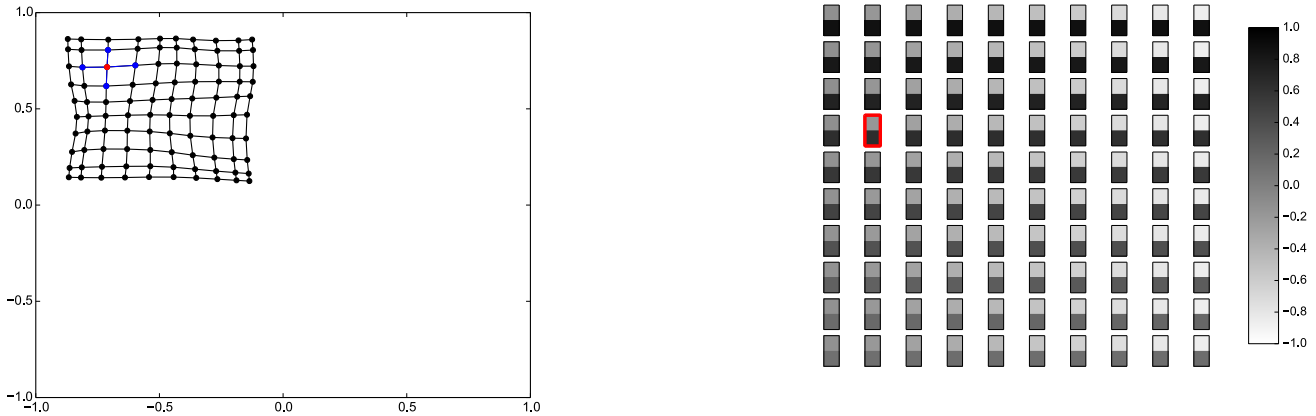


FIGURE 1 – Affichages obtenus après apprentissage sur le jeu de donnée 1

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. Par exemple, le neurone à la position (3,3) dans la carte a les poids représentés par le cercle rouge (ici [-0.6,0.7]) 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 [-0.2,0.8])

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 privilégiez 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 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.

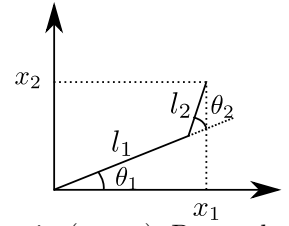
## 4.3 Analyse de l'algorithme (8 points + 2 points bonus)

- 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 plus la mesure d'auto-organisation que vous avez proposé à la section 3.2) :
  - taux d'apprentissage  $\eta$
  - largeur du voisinage  $\sigma$
  - 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 (4.5 points + 3 points bonus)

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.



On veut apprendre la relation entre la commande motrice  $(\theta_1, \theta_2)$  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.

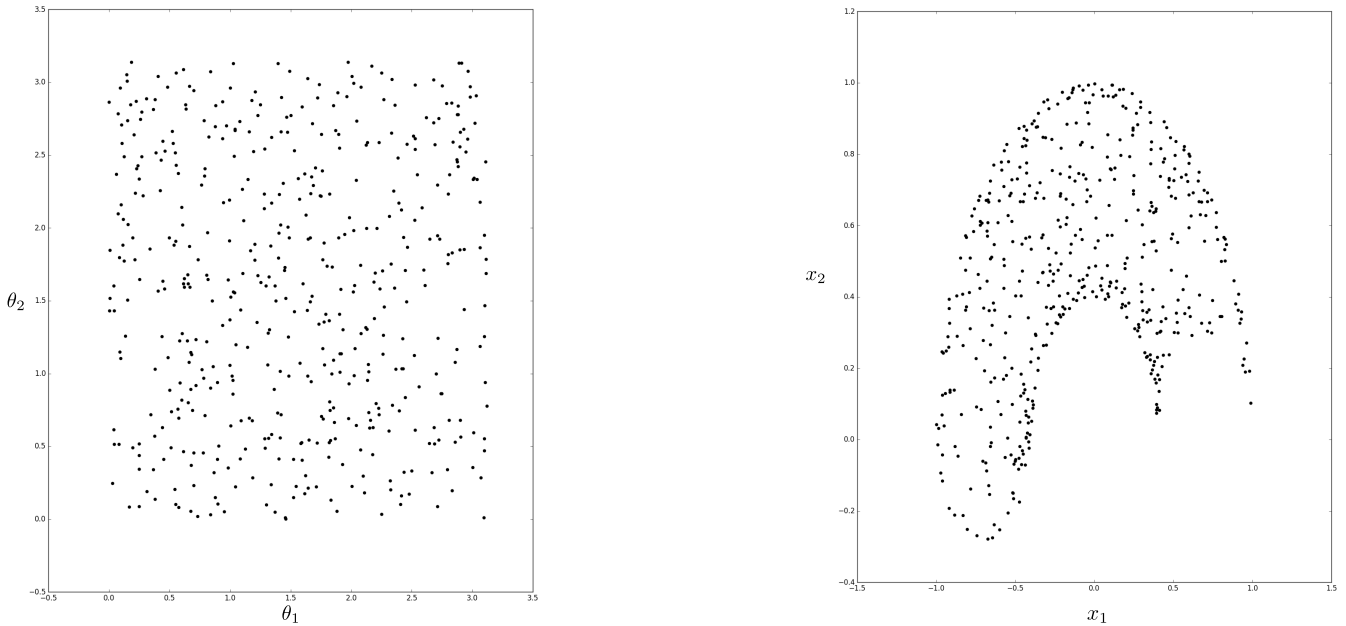


FIGURE 2 – Exemples de données de la position motrice du bras (à gauche) et de la position de la main dans l'espace (à droite), dans le cas où  $l_1 = 0.7$  et  $l_2 = 0.3$

Dans cette partie (hors partie bonus) on ne demande pas d'implémentation mais d'expliquer précisément comment vous procéderiez.

- 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 ?
- 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  $(\theta_1, \theta_2)$  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.
- (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).