**TP : Intelligence Artificielle**

**Réseaux de neurones**

**3. Étude “théorique” de cas simples**

3.1 Influence de η

On a la formule de mise à jour des poids pour chacune des connexions :

• Si l’on a cela implique donc = 0 ainsi il n’y a pas de variation dans ce cas-là.

• Dans le cas on a alors or pour le neurone gagnant, on a d’où => , le poids du neurone gagnant sera modifié pour s’approcher de l’entrée courante.

• Dans le cas où , le prochain point se rapprochera de X. Plus est proche de 0, moins ce poids sera proche de l’entrée courante.

• Si , le poids risque de dépasser notre point X.

3.2 Influence de σ

• Si σ augmente, augmente, donc augmente. Ainsi, si σ augmente, les neurones vont plus apprendre l’entrée courantes. Plus σ est grand, plus le voisinage est large, ce qui signifie que plus de neurones vont être impliqués dans la mise à jour des poids.

• Les neurones se rapprochant de X lorsque σ augmente, l’auto-organisation sera plus « resserrée », c’est-à dire distance plus faible entre les poids des neurones proches. Si σ est petit, les poids des neurones voisins seront moins affectés par les mises à jour, ce qui peut conduire à une auto-organisation plus lâche.

• On utilisera la fonction de variance :

3.3 Influence de la distribution d’entrée

• Si **X1** et **X2** sont représentés autant de fois, le vecteur de poids du neurone sera égal à la distance moyenne des deux entrées. .

• Si **X1** est présenté n fois plus que **X2**, alors le vecteur de poids de mesure sera n fois plus proche de **X1** que de **X2**.

• Les neurones tendent vers les zones de densité forte de données et seront donc plus espacés dans les zones où la densité de données est plus faible

**4. Étude pratique**

4.2 Implémentation

*Voir code (kohonen.py)*

4.3 Analyse de l’algorithme

Lorsque nous laissons les paramètres de base, voici les résultats que nous obtenons :

Une image contenant motif, texte, capture d’écran

Description générée automatiquementUne image contenant texte, capture d’écran, carré, Rectangle

Description générée automatiquementUne image contenant diagramme, ligne, texte, noir et blanc

Description générée automatiquement

**• Modification du taux d’apprentissage η**

Selon notre étude théorique, nous savons que le taux d'apprentissage η est un paramètre important de l'algorithme de Kohonen. Il contrôle la vitesse à laquelle les neurones de la carte s'ajustent aux données d'entrée. En l’augmentant, on remarque que les neurones apprennent beaucoup plus rapidement de l’entrée courante comme nous l’avons dit précédemment. Si η est trop grand, les poids des neurones peuvent osciller et l'apprentissage peut ne jamais converger. Si η est trop petit, l'apprentissage sera très lent. Nous supposons donc que le taux d'apprentissage aura un impact significatif sur les performances de l'algorithme de Kohonen.

Nous allons exécuter l’algorithme de Konohen avec différentes valeurs du taux d’apprentissage puis nous pourrons confronter nos résultats. Pour cela on fixe σ = 1.4, N = 30 000 et le réseau avec une entrée (2,1) et une carte (10,10).

Voyons comment le taux d’apprentissage influera sur notre réseau, et pour cela nous calculons l’erreur de quantification vectorielle et la variance.

|  |  |  |
| --- | --- | --- |
| Taux d’apprentissage η | Erreur de quantification vectorielle {X} | Variance |
| 0.01 | 0.025488703873810498 | 0.16035625678550 |
| 0.02 | 0.018239876182583593 | 0.16670077897012 |
| 0.05 | 0.028620886648823108 | 0.16529386294015 |
| 0.1 | 0.02575673957422963 | 0.14258732174433 |
| 0.2 | 0.01845569535843611 | 0.17695458804823 |
| 0.5 | 0.019094912573068657 | 0.16550255521239 |
| 1 | 0.030511079973037283 | 0.18626834774826 |
| 2 | 0.06504340508207526 | 0.26717742031518 |
| 5 | 0.0501600146754202 | 45.6535218327749 |

Une image contenant diagramme, ligne, capture d’écran, motif

Description générée automatiquementUne image contenant ligne, diagramme

Description générée automatiquement

η = 5

η = 0.01

On remarque que {X} garde des valeurs à peu près constantes et il ne semble pas y avoir de valeur idéale du taux d’apprentissage minimisant {X}, hormis pour des très faibles de η. Il en va de même pour la variance dont la variation est très faible selon la valeur de η. Au-dessus d’un taux égal à 1, l’erreur de quantification vectorielle augmente. Aussi la variance augmente de manière exponentielle. Ainsi un taux d’apprentissage inférieur ou égal à 1 est préférable. Jetons un œil aux poids des neurones après apprentissage :

Une image contenant texte, logiciel, Logiciel multimédia, Site web

Description générée automatiquementUne image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Description générée automatiquementUne image contenant texte, logiciel, Logiciel multimédia, Site web

Description générée automatiquement

η = 0.99

η = 0.5

η = 0.01

On remarque cette fois que plus η est grand, plus les poids se rapprochent du point d’apprentissage.

Ces résultats confirment nos précédentes hypothèses :

• Plus le taux d’apprentissage est proche de 1, plus l’apprentissage des neurones est rapide.

• Un taux d’apprentissage trop grand engendre la divergence de l’apprentissage.

• une trop faible valeur de η aide moins à l’apprentissage.

**• Modification de la largeur du voisinage σ**

Lors de l’étude théorique, nous avons avancé que σ influait également sur la façon dont les neurones vont apprendre de l’entrée courante. Ainsi plus σ est grand plus l’auto-organisation serait globale. Également, une haute valeur de σ pourrit engendrer à une meilleure quantification vectorielle et moins de cluster entre les neurones de poids semblables.

Nous allons cette fois-ci exécuter l’algorithme de Konohen avec différentes valeurs de la largeur du voisinage puis nous pourrons confronter nos résultats. Pour cela on fixe η = 0.05, N = 30 000 et le réseau avec une entrée (2,1) et une carte (10,10).

|  |  |  |
| --- | --- | --- |
| Largeur du voisinage σ | Erreur de quantification vectorielle {X} | Variance |
| 0.01 | 0.012840091383675561 | 0.22467493948832 |
| 0.05 | 0.013179449510584359 | 0.22619486841755942 |
| 0.1 | 0.014685672310431525 | 0.24047839985771477 |
| 0.5 | 0.009194948975820103 | 0.18730256399755837 |
| 1 | 0.014722101813487477 | 0.18174642204900038 |
| 1.4 | 0.028620886648823108 | 0.16529386294015 |
| 2 | 0.05407709653114227 | 0.13908697833629394 |
| 3 | 0.09547645581933438 | 0.08690878968669397 |
| 5 | 0.18508551872509613 | 0.035326736120438226 |

Une image contenant capture d’écran, ligne, diagramme

Description générée automatiquementUne image contenant texte, capture d’écran, ligne, Rectangle

Description générée automatiquement

σ = 5

σ = 0.01

Comme nous l’avons prédit, une grande valeur de σ favorise la formation de neurones ayant des poids similaires, si σ est petit (voir σ=0.01), le réseau est moins organisé. On remarque également ce résultat avec la variance qui est inversement proportionnelle à σ. Plus σ est grand, plus la variance sera petite et donc le réseau auto-organisé. {X} ne semble pas être particulièrement influencé par σ, hormis lorsque nous avons des valeurs de σ grandes, cette valeur est plus importante.

Par exemple, si l’on prend σ=1000 on remarque qu’il n’y a plus de distance entre les neurones et qu’ils sont tous collés. **Une image contenant texte, capture d’écran, diagramme, ligne

Description générée automatiquement**

**• Modification du taux d’apprentissage N**

Plus le nombre de pas d'apprentissage est élevé, plus l'algorithme aura le temps de converger vers une solution optimale.

N représente le nombre d’itérations, le nombre de fois que l’algorithme va parcourir l’ensemble des données d’apprentissage pour ajuster les poids des neurones.

Plus ce taux d’apprentissage serait élevé, plus il devrait mener notre réseau à converger vers une solution optimale par définition.

On fixe η = 0.05 et σ = 1.4 et le réseau avec une entrée (2,1) et une carte (10,10).

|  |  |  |
| --- | --- | --- |
| Nombre de pas du taux d’apprentissage N | Erreur de quantification vectorielle {X} | Variance |
| 100 | 0.13359195007967434 | 0.07538932963698763 |
| 500 | 0.05162688393463655 | 0.14033566158765123 |
| 2000 | 0.028706617351032058 | 0.1593932337681165 |
| 10000 | 0.026765133670947443 | 0.1533005076837436 |
| 30000 | 0.028620886648823108 | 0.16529386294015 |
| 80000 | 0. 023754932663695858 | 0. 1627678252589603 |
| 150000 | 0.018748897406778427 | 0.16375768733367374 |
| 500000 | 0.02382873279861369 | 0.15303081191575002 |

Lorsque l’on modifie la taille et forme de la carte (entrée et nombre de neurones), cela devrait modifier l’auto-organisation des neurones. Une carte de taille importante permettrait une meilleure représentation de l’espace d’entrée mais cela devrait provoquer une plus longue durée pour l’apprentissage des neurones.

Une image contenant texte, croquis, capture d’écran, dessin

Description générée automatiquementUne image contenant diagramme, ligne, origami, motif

Description générée automatiquement  
  
  
  
En effet en augmentant N, on retrouve plus de points qui rejoignent le point d’apprentissage.

N =500000

N =100

Ainsi, en améliorant N cela permet une exploration plus approfondie de l’espace de recherche des poids de neurones. Cependant l’erreur de quantification vectorielle finit par stagner, cela est dû au fait que le poids des aurones aura déjà converger vers une solution optimale. La variance est également assez stable sauf pour des trop petites valeurs de N.

**• Modification de la taille et la forme de la carte**

**Une image contenant texte, diagramme, capture d’écran, ligne

Description générée automatiquementUne image contenant ligne, diagramme, capture d’écran, Tracé

Description générée automatiquement**

Comme on le voit, en changeant la carte et la forme, l'auto-organisation des neurones est totalement différente. Ce qui veut dire que les poids vont facilement se rapprocher de la forme adéquate.

Une carte de grande taille offre également une meilleure représentation de l’espace d’entrée mais nécessitera plus de temps d’apprentissage. Évidemment l’impact de la taille et la forme de la carte sur l’algorithme de Konohen dépendra également du jeu de données avec lequel nous travaillons ainsi que les autres paramètres étudiés précédemment.

**• Modification du jeu de données**

Les données peuvent permettre de mieux comprendre la capacité de l'algorithme à trouver une structure dans des données complexes.

Le premier jeu de données étaient distribués uniformément dans le quadrant [−1, 1] × [−1, 1]

On essaie avec le deuxième jeu de données distribués uniformément dans les quadrants

[−1, 0] × [0, 1], [−1, 0] × [−1, 0] et [0, 1] × [−1, 0] :

**Une image contenant diagramme, ligne, texte, capture d’écran

Description générée automatiquement**

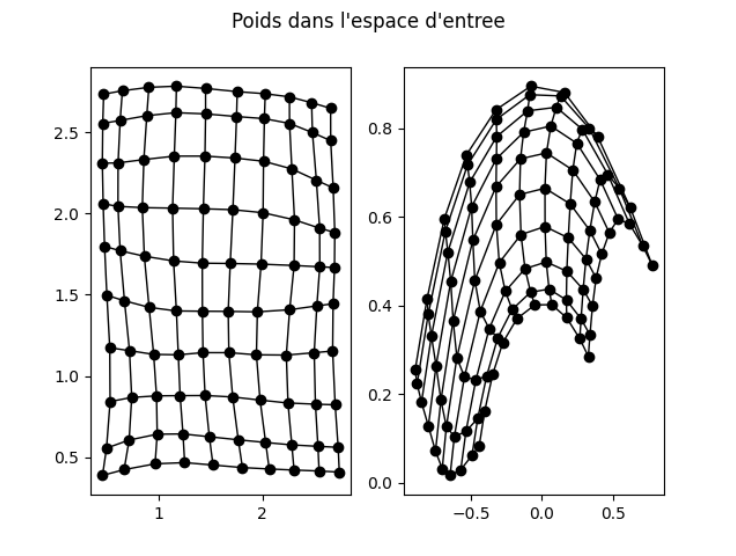
Désormais avec le jeu de données distribués uniformément dans les quadrants [−1, 0] × [0, 1] et [0, 1] × [−1, 0] :

**Une image contenant capture d’écran, texte, diagramme, ligne

Description générée automatiquement**

Nos jeux de données ont des distributions différentes, le premier est généré aléatoirement et est uniforme, il n’a pas une structure particulière mais lorsque l’algorithme s’applique les données se séparent de manière assez homogène.  
  
Le deuxième jeu de données (également généré aléatoirement) mais avec trois groupes de points, ainsi l’algorithme se base sur un jeu de données plus complexe.

4.4 Bras robotique



• Une fois que la carte de Kohonen a été entraînée, nous pouvons utiliser cette carte pour prédire la position spatiale du bras robotique en fonction d'une position motrice donnée. De même, nous pouvons prédire la position motrice correspondant à une position spatiale souhaitée.

Pour prédire la position spatiale à partir d'une position motrice, nous utilisons la carte de Kohonen qui a appris à associer les positions motrices aux positions spatiales. Nous calculons l'activité des neurones de la carte en utilisant la position motrice donnée comme entrée. Ensuite, nous trouvons le neurone gagnant, c'est-à-dire le neurone dont le poids est le plus proche de l'entrée motrice. Les valeurs spatiales associées à ce neurone gagnant sont utilisées comme prédiction de la position spatiale correspondante.

De manière similaire, pour prédire la position motrice à partir d'une position spatiale souhaitée, nous utilisons la carte de Kohonen. Nous calculons l'activité des neurones de la carte en utilisant la position spatiale souhaitée comme entrée. Ensuite, nous trouvons le neurone gagnant, c'est-à-dire le neurone dont le poids est le plus proche de l'entrée spatiale. Les valeurs motrices associées à ce neurone gagnant sont utilisées comme prédiction de la position motrice correspondante.

Pour implémenter cela, nous pouvons utiliser la fonction `compute` de l’algorithme de Kohonen pour calculer l'activité des neurones, puis trouver le neurone gagnant en utilisant les coordonnées du neurone avec la plus faible activité. Enfin, nous utilisons les valeurs spatiales ou motrices associées à ce neurone gagnant comme prédiction.

**Pour l’implémentation des fonctions, voir le fichier *BrasRobotique.py* *(fonctions predictSpatialPosition et predictMotorPosition)* associées.**

• Le modèle Hopfield se rapproche de cette méthode (apprentissage sur le quadruplet pour ensuite en retrouver une sous partie étant donnée l’autre).

Le modèle Hopfield présente des similitudes avec la méthode de l'apprentissage sur le quadruplet utilisé dans la carte de Kohonen pour prédire une sous-partie étant donnée l'autre. Dans le modèle Hopfield, les connexions entre les neurones sont utilisées pour stocker des modèles ou des motifs spécifiques, et ces modèles peuvent être récupérés en fournissant une partie du modèle en entrée.

|  |  |
| --- | --- |
| Avantages | Inconvénients |
| Réseau auto-associatif (Permet d’associer des modèles ou des motifs à des représentations internes du réseau) | Sensibilité aux perturbations (sensible aux perturbations et aux bruits) |
| Capacité de stockage (Peut stocker un nombre important de motifs grâce à ses connexions récurrentes) | Limitation de cette capacité (Capacité de stockage élevée mais limite pratique sur le nombre de motifs pouvant être stockés sans interférences) |
| Dynamique asynchrone (Les neurones peuvent être mis à jour séquentiellement plutôt que simultanément) | Dépendance aux modèles d’entrainement (dépend des modèles d’entrainement spécifiques ayant été utilisés pour le stockage initial des motifs) |

• Si l'objectif était de prédire uniquement la position spatiale à partir de la position motrice, le modèle Perceptron pourrait être utilisé. Voici les avantages et les inconvénients associés à l'utilisation du modèle Perceptron dans ce contexte :

|  |  |
| --- | --- |
| Avantages | Inconvénients |
| Simplicité (Compréhension et utilisation) | Limitation de la linéarité (Limité aux tâches de classification linéaire) |
| Adaptabilité (Peut résoudre des problèmes de classification linéaire) | Représentation limitée (Limité aux fonctions linéaires) |
| Apprentissage supervisé (Peut être entraîné à prédire la position spatiale à partir de la position motrice à l’aide d’un ensemble de données d’entrainement approprié) | Sensibilité aux valeurs aberrantes (Ou aux données bruitées) |

• Pour prédire la suite des positions spatiales prises par la main lors du déplacement du bras d'une position motrice (θ1, θ2) à une nouvelle position (θ1', θ2'), nous pouvons utiliser la carte de Kohonen apprise. Voici le principe pour prédire la trajectoire :

1. Calculer la position spatiale correspondant à la position motrice initiale (θ1, θ2) en utilisant la carte de Kohonen. Cela peut être fait en utilisant la méthode de prédiction de la position spatiale à partir de la position motrice que nous avons discutée précédemment.

2. Utiliser la nouvelle position motrice (θ1', θ2') pour prédire la nouvelle position spatiale en utilisant la carte de Kohonen, l’enregistrer.

3. Mettre à jour la position motrice actuelle (θ1, θ2) avec la nouvelle position motrice (θ1', θ2').

Et l’on répète les étapes jusqu’à ce que la position motrice atteigne la nouvelle position motrice souhaitée (θ1', θ2').

**Pour l’implémentation des fonctions, voir le fichier *BrasRobotique.py***