Devoir 5 : Réseaux de Hopfield

PHQ404

Date de complétion : 19 avril 2024 à 23h45

1 Objectif

L'objectif de ce devoir est d'implémenter un réseau de Hopfield.

2 Introduction

John Hopfield (1933-) est un physicien et biologiste théorique américain dont les contributions ont profondément influencé le domaine de l'intelligence artificielle et de la neurosciences computationnelles. Il est particulièrement célèbre pour avoir introduit en 1982 le réseau de Hopfield, un type de réseau de neurones artificiels avec une capacité de mémoire associative, capable de servir comme système de récupération de l'information avec correction d'erreur. Le modèle de Hopfield, utilisant une dynamique énergétique pour atteindre des états stables qui représentent la mémoire. Ce modèle a fourni un mécanisme pour comprendre comment les réseaux de neurones peuvent auto-organiser l'information et a inspiré de nombreuses recherches sur les algorithmes d'optimisation et les modèles de mémoire dans le cerveau. Les travaux de Hopfield ont également contribué à établir des ponts entre la physique et la biologie computationnelle, enrichissant la compréhension des systèmes complexes et du calcul neuronal. Sa vision interdisciplinaire a jeté les bases de nombreuses avancées dans l'étude des systèmes complexes, tant biologiques qu'artificiels.

Articles fondateurs :

- J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities.," *Proceedings of the national academy of sciences*, t. 79, n° 8, p. 2554-2558, 1982 [1]
- J. J. Hopfield, "Neurons with graded response have collective computational properties like those of two-state neurons.," *Proceedings of the national academy of sciences*, t. 81, n° 10, p. 3088-3092, 1984 [2]

Le réseau de Hopfield est un réseau de neurones de type McCullogh et Pitts tel que :

- Le réseau est **complet et simple** : chaque neurone est connecté à tous les autres neurones sauf à lui-même. Il est donc récurrent mais sans boucle.
- De plus, les poids de connexions sont **symétriques**, c'est-à-dire $W = W^\top$ et donc $w_{ij} = w_{ji}$ pour tous les i, j.

- Les poids de connexion peuvent être négatifs, ce qui signifie que le réseau peut avoir de l'**inhibition**, c'est-à-dire le processus neurologique par lequel une cellule nerveuse (neurone) réduit l'activité d'une autre cellule nerveuse, limitant ainsi ou empêchant la transmission de signaux excitateurs.
- Les poids de connexion sont plastiques et changent selon une **règle de Hebb**, souvent résumée par l'adage "ce qui se renforce ensemble, se connecte ensemble", est un principe fondamental en neurosciences et en théorie des réseaux de neurones artificiels qui décrit comment les connexions entre les neurones du cerveau se renforcent durant l'apprentissage. Formulée en 1949 par **Donald Hebb** (1904-1985), elle stipule que si deux neurones sur les deux côtés d'une synapse (connexion) sont activés simultanément, alors la force de cette connexion augmente. En d'autres termes, la répétition de l'activation simultanée renforce la tendance de ces neurones à s'activer ensemble à l'avenir.

2.1 Dynamique d'un réseau de Hopfield

- **Activité binaire** : l'activité du neurone i à l'instant t, $x_i(t)$, est égale à 0 ou 1.
- **Temps discret** : t appartient à un ensemble dénombrable (discret) et pour simplifier on suppose que $t \in \{0, 1, 2, \ldots\}$.
- **Mise à jour asynchrone** : À tout instant t, l'activité d'un et d'un seul neurone peut être modifiée. L'indice du neurone mis à jour est choisi aléatoirement à tout instant t.
- **Règle de mise à jour** : Si le neurone i à l'instant t est mis à jour , alors

$$x_i(t+1) = H\left(\sum_{j=1}^{n} w_{ij} x_j(t) - \theta_i\right),$$

ce qui signifie que

$$x_i(t+1) = \begin{cases} 1 & \text{si } \sum_{j=1}^{n} w_{ij} x_j(t) > \theta_i, \\ 0 & \text{si } \sum_{j=1}^{n} w_{ij} x_j(t) \leq \theta_i. \end{cases}$$

Sous forme vectorielle,

$$\mathbf{x}(t+1) = H\left(W\mathbf{x}(t) - \boldsymbol{\theta}\right).$$

2.2 Quelques concepts fondamentaux

- **État stationnaire** : si $\mathbf{x}(t) = \mathbf{x}^*$ pour tout t, alors $\mathbf{x}^*$ est un état stationnaire.
- **Fonction d'énergie** (Lyapunov) :

$$E(\mathbf{x}) = -\frac{1}{2}\sum_{i \neq j} w_{ij} x_i x_j + \sum_i \theta_i x_i = -\frac{1}{2}\mathbf{x}^\top W \mathbf{x} + \boldsymbol{\theta}^\top \mathbf{x}.$$

- **Résultat important de Hopfield** : avec la mise à jour asynchrone et les connexions symétriques,

$$E\big(\mathbf{x}(t+1)\big) \leq E\big(\mathbf{x}(t)\big),$$

ce qui signifie que les minima locaux de E sont des états stationnaires stables.
- **Mémoire** : un état stationnaire est un état mémorisé. En partant d'un état proche, on peut converger au fil du temps vers cet état ou vers un autre état stationnaire.

2.3 Plasticité du réseau de Hopfield : apprentissage d'états

- **Apprentissage d'un état** : pour stocker un seul état $\mathbf{y}$, on change $W_{i,j}$ pour

$$W_{ij} + \begin{cases} (2y_i - 1)(2y_j - 1) & \text{si} \quad i \neq j, \\ 0 & \text{si} \quad i = j. \end{cases}$$

- **Apprentissage de plusieurs états** : pour stocker les $\mathbf{y}^\mu$, $\mu = 1, \ldots, p$, on change $W_{i,j}$ pour

$$W_{ij} + \begin{cases} \sum_\mu (2y_i^\mu - 1)(2y_j^\mu - 1) & \text{si} \quad i \neq j, \\ 0 & \text{si} \quad i = j. \end{cases}$$

- **Lien avec Hebb** : les termes $y_i^\mu y_j^\mu$ ajoutent du poids à la connexion (i, j) si les neurones i et j sont activés quand le réseau est dans l'état $\mathbf{y}^\mu$.

3 Comment présenter et remettre votre TP

Vous devez cloner le répertoire github dans l'organisation du cours au lien suivant : https://classroom.github.com/a/FJR5qORQ. Dans ce répertoire se trouvera votre code python, vos tests unitaires ainsi que votre rapport décrivant les méthodes utilisés et l'analyse de vos résultats. La structure des fichiers ne doit pas être modifiée, mais vous pouvez ajouter des fichiers si vous le désirez. Voici la structure de fichiers que votre répertoire devra garder :

Root
- → src
  - ↪ `fichier0.py`
  - ↪ `fichier1.py`
  - ↪ ...
- → tests
  - ↪ `test_fichier0.py`
  - ↪ `test_fichier1.py`
  - ↪ ...

↪ `.gitignore`
↪ `requirements.txt`
↪ `README.md`

Le fichier `requirements.txt` doit contenir les dépendances de votre projet. Le fichier `README.md` doit contenir les instructions pour installer et utiliser votre projet ainsi qu'une brève description du devoir et des méthodes utilisés dans le code. Voir la section 6 pour plus de détails. Dans le dossier `src` se trouvera votre code python et dans le dossier `tests` se trouvera vos tests unitaires.

La remise et la correction automatique du code se fera à chaque `push` sur le répertoire github. Notez que seul le dernier `push` sur la branche `main` sera considéré pour la correction.

4 Énoncé

4.1 Modèle de Hopfield

Vous allez devoir implémenter une classe `HopfieldNetwork` qui permet de créer un réseau de Hopfield. Dans celle-ci, vous devrez implémenter les méthodes suivantes :

- `__init__` : le constructeur de la classe qui prend en argument les poids de connexions `weights`, les seuils `thresholds` et l'état initial `initial_state` du réseau.
- `n` : Une propriété qui retourne le nombre de neurones dans le réseau.
- `energy` : Une méthode qui retourne l'énergie du réseau.
- `get_state_energy` : Une méthode qui calcule l'énergie d'un état donné.
- `set_state` : Une méthode qui permet de changer l'état du réseau.
- `update` : Une méthode qui met à jour l'état du réseau de manière asynchrone sur un pas de temps.
- `simulate` : Une méthode qui met à jour l'état du réseau de manière asynchrone sur un nombre de pas de temps `m` et qui retourne l'historique des états du réseau.
- `learn_one_state` : Une méthode qui permet d'apprendre un état `state` en modifiant les poids de connexions.
- `learn_many_states` : Une méthode qui permet d'apprendre plusieurs états `states` en modifiant les poids de connexions.

4.2 Visualisation des résultats

Vous allez devoir présenter les graphiques suivants :

- L'énergie du réseau en fonction du temps pour des états mémorisés différents.
- L'évolution de l'état du réseau en fonction du temps pour des états mémorisés différents. (Utilisez des gifs)

Vous allez devoir faire ces derniers graphiques pour les états 'image' et 'message' qui sont dans le dossier `tests/data`. En plus, vous allez devoir ajouter une petite analyse sur l'apprentissage de plusieurs états (fait avec la méthode `learn_many_states`) et sur la dynamique du réseau pour ces états.

Note : Toutes les fonctions qui doivent être implémentées sont déjà définies dans les fichiers et retournent des `NotImplementedError`.

5 Vérification

Il est important de vérifer vos implémentations. En effet, vous devez vous assurer que vos méthodes fonctionnent correctement et pour ce faire, vous devez rouler et implémenter des tests unitaires qui testent chacune de vos classes et fonctions. De plus, vous devriez tester si les résultats obtenus sont logiques. Il serait aussi intéressant de retrouver vos vérifications dans votre rapport. Il est fortement recommandé d'ajouter des tests unitaires dans le dossier `tests`, mais les tests déjà implémentés ne doivent pas être modifiés.

6 Readme

Vous devez faire un fichier Readme qui explique ce que contient votre répertoire et comment l'utiliser. Le Readme sera divisé en 2 parties : une partie plus courte qui consiste essentiellement à ce qu'on retrouve normalement dans Readme scientifique et une partie plus longue qui consiste en une présentation et analyse des résultats.

La première partie doit contenir les éléments suivants :

- Une brève description du contenue du répertoire ;
- Une figure qui résume le contenu du répertoire ainsi que les résultats principaux ;
- Les instructions pour installer et utiliser votre projet.

Il faut qu'un utilisateur externe soit en mesure de regarder la première partie du Readme et comprendre en quelques secondes le contenu du répertoire, les résultats principaux et comment utiliser le projet. C'est important d'être concis, clair et efficace. Pour la figure, il sagit d'une image permettant au lecteur de comprendre rapidement le contenu du répertoire. Celle-ci pourrait être, par exemple, un diagramme représentant le pipeline de traitement des données, un graphique comparant les différentes méthodes implémentées, etc.

Le deuxième partie doit contenir les éléments suivants :

- Une plus longue description du contenu du répertoire ;
- Une présentation et explication des méthodes utilisées ;
- Une présentation des résultats obtenus ;
- Une analyse des résultats obtenus ;
- Une conclusion.

Cette seconde partie sert à expliquer en quoi consiste ce dépôt si l'utilisateur décidait que la première partie du Readme était assez intéressante et bien présentée pour qu'il veuille en savoir plus. Il sagit ici d'un court rapport scientifique. Il faut donc rester concis afin d'être lu en quelques minutes seulement, mais mettre suffisamment d'information pour que l'utilisateur comprenne bien la théorie, les méthodes et les résultats.

7 Critères d'évaluation

70 points Pour le résultat de l'autocorrection du code obtenue à l'aide du module TAC.

30 points Pour la qualité du Readme.

Références

[1] J. J. HOPFIELD, "Neural networks and physical systems with emergent collective computational abilities.," *Proceedings of the national academy of sciences*, t. 79, n° 8, p. 2554-2558, 1982.

[2] J. J. HOPFIELD, "Neurons with graded response have collective computational properties like those of two-state neurons.," *Proceedings of the national academy of sciences*, t. 81, n° 10, p. 3088-3092, 1984.