MASTER EN SCIENCES MATHÉMATIQUES

Evolution spontanée de la modularité dans un modèle simple de reconnaissance de motifs

Namur, Aurore

Award date:
2018

Awarding institution:
Universite de Namur

Remerciements

Je voudrais remercier les différents membres de mon jury et en particulier mon promoteur Monsieur Carletti ainsi que Delphine pour leurs relectures de ce mémoire et leurs conseils.

Je tiens également à remercier mon frère Jean-Sébastien pour l'aide qu'il m'a apportée sur le fonctionnement du système nerveux chez l'être humain.

Merci également aux professeurs et aux assistants qui m'ont suivie durant ces cinq dernières années. Je voudrais également remercier les étudiants de mon année pour leur soutien, leur amitié et leur bonne humeur.

Pour terminer, je souhaite remercier toute ma famille et mes amis qui m'ont soutenue durant ces cinq ans d'étude.

Résumé

Les réseaux de neurones artificiels sont constitués d'une couche d'entrées, de couches cachées et d'une couche de sortie. Des signaux sont perçus par les neurones de la couche d'entrées. Le réseaux fournit ensuite une réponse par les neurones de la couche de sortie en fonction des connexions avec les neurones intermédiaires.

Un réseau de neurones artificiels peut avoir une structure modulaire. Cela signifie que les neurones qui le composent peuvent être divisés en communautés. Dans ces réseaux les connexions inter-communautaires sont peu nombreuses tandis que celles intra-communautaires sont denses. Afin de quantifier cette notion, la mesure de Newman et Girvan est utilisée.

Dans ce mémoire, les réseaux de neurones sont évolués à l'aide d'un algorithme génétique afin d'atteindre une tâche d'apprentissage. Cette tâche est la reconnaissance de motifs provenant d'une rétine gauche et d'une rétine droite. Les réseaux de neurones et les algorithmes génétiques sont tout d'abord étudiés de manière générale. Après cela, une description du problème étudié est donnée et les différents concepts, tels que les réseaux de neurones et les algorithmes génétiques, sont appliqués au problème. Une implémentation du problème est ensuite expérimentée. Les différents paramètres utilisés pour l'implémentation de l'algorithme génétique ainsi que la représentation des réseaux de neurones sont tirés de l'article de N. Kashtan et U. Alon [1]. Lors de l'expérimentation, les réseaux sont, dans un premier temps évolués dans le but d'atteindre une tâche fixe. Dans un second temps, ils sont évolués en intervertissant entre deux objectifs qui sont une combinaison différente de mêmes sous-buts. La modularité des différents réseaux obtenus est ensuite calculée avant d'être comparée avec les résultats de l'article.

Mots clés : réseaux de neurones artificiels, modularité, algorithme génétique, mutation, croisement, reconnaissance de motifs.

Abstract

Artificial Neural Networks are formed of an entry layer, hidden layers and an output layer. Signals are picked up by neurons from the entry layer. Then networks provide an answer through neurons from the output layer depending of the connections with the intermediate neurons.

An artificial neural network can have a modular structure. This means that neurons constituting this network can be split into communities. Within these networks, the quantity of inter-community connections is limited while the density of intra-community connections is high. To quantify this concept, a measure based on the approach of Newman and Girvan is used.

In this master's essay, neural networks are evolved by means of a genetic algorithm to reach a learning task. This task is the recognition of shapes coming from a right retina and from a left retina. At first, neural networks and genetic algorithms are globally studied. After that, the problem to be analyzed is described and the different concepts like the neural networks and the genetic algorithms are applied to solve the problem. An implementation of the problem is then tested out. The different parameters used for the genetic algorithm and the representation of the neural networks are drawn from N. Kashtan and U. Alon's article [1]. During the experiment, networks are first evolved to obtain a set task. In a second step, networks are evaluated by switching between two goals, each made of a different combination of subgoals. The modularity of the different networks obtained is computed and compared with the results of the article.

Keywords : artificial neural networks, modularity, genetic algorithm, mutation, crossover, pattern recognition.

Table des matières

Introduction

Chez l'être humain, le cerveau est composé de neurones. Ces derniers interagissent être eux et forment un réseau de communication complexe. Selon la théorie de Hebb [2], lorsque nous voyons une image, un visage,... ou encore lorsque nous répétons un mouvement, cela stimule un groupe de neurones dépendant du stimulus initial. Plus ce stimulus est important et répéter, plus les connexions dans ce groupe de neurones sont renforcées et importantes. De tel groupes sont appelés communautés ou modules. Les connexions dans ces communautés sont donc importantes tandis que celles entre ces groupes sont moindres. Les réseaux présentant une telle structure sont dits modulaires.

Les réseaux de neurones artificiels s'inspirent de ces réseaux biologiques dans le but, par exemple, de reconnaître des schémas. Mais actuellement, cette structure modulaire n'est pas, ou très peu, présente dans les modèles de l'évolution biologique. La plupart de ces modèles font évoluer les réseaux afin d'atteindre un but bien défini et ne présentent pas de structure divisée en modules. Cela est dû au fait que cette structure est généralement moins optimale. Ainsi, même si une solution est initialement modulaire, elle va rapidement évoluer vers une solution qui ne l'est pas.

Dans l'article [1] de N. Kashtan et U. Alon, les auteurs donnent une explication possible de l'origine de la modularité dans les réseaux de neurones biologiques. Pour ce faire, ils font évoluer leurs réseaux dans un environnement qui change au cours du temps. En effet, cet environnement se modifie afin d'atteindre deux tâches d'apprentissage. Ces tâches ont pour caractéristique d'être des combinaisons différentes de mêmes sous-buts. Par exemple, une tâche pourrait être d'atteindre l'objectif A et B et une autre, A ou B.

Dans ce mémoire, nous nous concentrerons sur une expérience de reconnaissance de schémas. Pour ce faire, nous utiliserons des réseaux de neurones artificiels. L'objectif est ainsi d'essayer de reproduire les résultats de l'article. Afin de faciliter la compréhension du problème, nous commencerons par définir différents outils, à savoir les réseaux de neurones, les algorithmes génétiques (grâce auxquels nous ferons évoluer notre réseau) et la modularité. Ensuite, nous appliquerons ces notions à notre problème et nous analyserons et discuterons les résultats obtenus.

Chapitre 1

Réseaux de neurones

Dans ce chapitre, nous allons poser les bases biologiques d'un réseau de neurones afin de pouvoir ensuite le modéliser. Pour ce faire, nous développerons, à l'aide du livre du Dr A. Roberts [3] ce qu'est un neurone et ses différentes caractéristiques du point de vue de la biologie. Nous développerons ensuite la notion de neurones et réseaux artificiels, ainsi que celle de modularité d'un réseau.

1.1 Neurones biologiques

Le corps humain est constitué de millions de millions de cellules possédant des fonctionnalités différentes. Mais malgré leur diversité, certains attributs peuvent ressortir de leur structure. En effet toute cellule possède un centre de commandement, appelé noyau, dans lequel se trouve son ADN, une membrane plasmique maintenant sa forme et contrôlant les entrées et sorties, et des mitochondries apportant l'énergie nécessaire au fonctionnement de la cellule. Le système nerveux comporte, quant à lui, des milliards de cellules qui communiquent entre elles. Ces dernières sont appelées cellules nerveuses ou bien encore neurones et comptent parmi les plus spécialisées du corps humain.

Il existe différents types et structures de neurones. Mais ils peuvent, de manière générale, être représentés tel qu'à la figure 1.1. Les filaments émergeant du corps cellulaire sont appelés dendrites. Ce sont elles qui reçoivent les charges électriques. Le corps cellulaire combine ensuite tous ces signaux pour envoyer son propre signal à d'autres cellules. Pour ce faire, cet influx est propagé, à partir du corps cellulaire jusqu'aux synapses, par un autre filament plus fin et plus long que les dendrites (pouvant aller jusqu'à 1 m de long), appelé axone. Un tel neurone peut communiquer avec des centaines d'autres cellules nerveuses et ainsi créer un formidable réseau de communication.

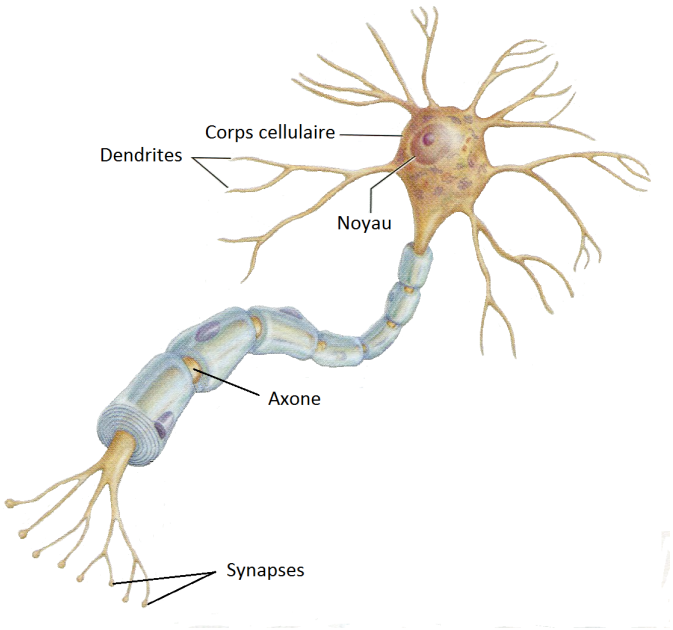


FIGURE 1.1: Représentation d'un neurone biologique. Source : [4]

Pour revenir aux influx nerveux, ceux-ci sont des signaux électriques. En effet, à l'extérieur et l'intérieur de la cellule se trouvent des liquides organiques qui sont électriquement neutres. Dans ces liquides, des minéraux tels que le sodium et le potassium passent à l'état d'ions, leur mouvement provoquant les impulsions électriques ou potentiel d'action des signaux. Les deux faces de la membrane ont des charges polarisantes, ce qui crée un potentiel de repos. Mais lorsque les ions traversent la membrane, cela provoque un déplacement des charges électriques et déclenche une impulsion comme illustré à la figure 1.2. Ces influx sont de différents types. Certains sont excitateurs (dépolarisants) tandis que d'autres sont inhibiteurs (hyperpolarisants). Les premiers contribuent donc à la transmission du message alors que les derniers empêchent la formation de signal dans le récepteur. L'influx nerveux déclenchera un potentiel d'action si la somme des influx excitateurs et inhibiteurs dépasse un certain seuil.

De plus, une faculté extraordinaire que possède un neurone est d'établir de nouvelles connexions avec d'autres neurones. Cela porte le nom de plasticité neuronale. En effet, l'apprentissage et la mémorisation créent de nouvelles connexions entre neurones. Ainsi lorsqu'un stimulus excite un neurone, celui-ci envoie une série d'impulsions au neurone suivant. Lorsque le stimulus revient régulièrement, cela renforce cette connexion initiale et peut éventuellement mobiliser des neurones supplémentaires. Par ailleurs, l'usage régulier du stimulus va maintenir l'existence et renforcer le signal de la connexion qui peut ainsi perdurer des semaines, des mois voir même des années. Dans le cas contraire, le lien va tendre à disparaître.

Cela a pour conséquence qu'un réseau de neurones biologique évolue dans le temps. Ce phénomène de plasticité neuronale est présent dès le développement embryonnaire, puis lors de l'enfance et de l'âge adulte. Cependant, plus une personne est jeune plus ce phénomène est important. Il est toujours possible à l'âge adulte mais beaucoup moins fortement.

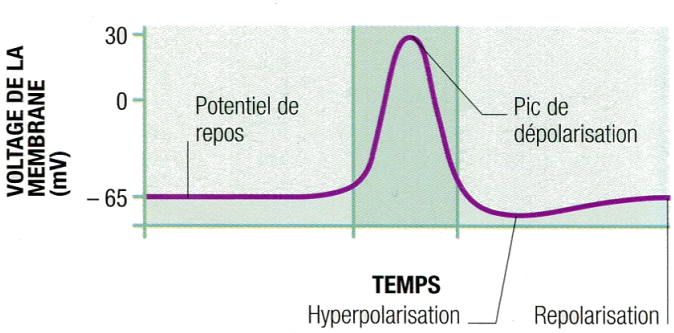


FIGURE 1.2: Potentiel d'action. Les ions entrent et sortent par des canaux ioniques de la membrane axonale, générant ainsi un potentiel d'action qui change le voltage de la cellule. Source : [3]

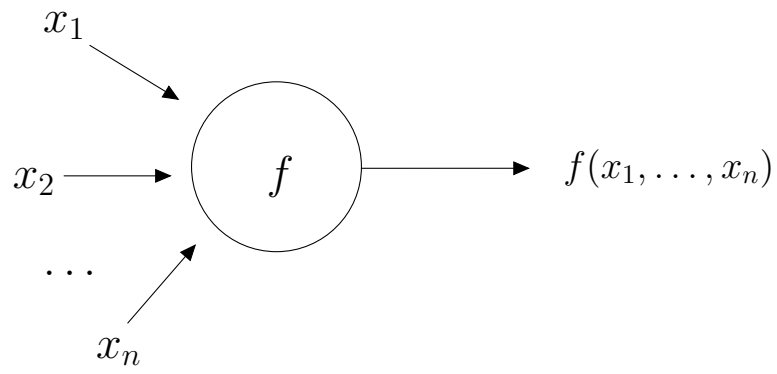
1.2 Neurones artificiels

Sur base de la référence [5] et de ce que nous avons vu à la section précédente, c'est-à-dire les différentes caractéristiques et composantes d'un neurone biologique, nous allons maintenant définir les neurones artificiels à l'aide d'un modèle appelé modèle du perceptron. Pour ce faire, nous retiendrons des neurones biologiques les quatre composantes suivantes :

- les dendrites,
- le corps du neurone,
- l'axone,
- les synapses.

Une première modélisation d'un neurone artificiel est illustré à la figure 1.3 (a). Ce neurone posséde n canaux d'entrées $x_1, ..., x_n$ et sa réponse est représentée comme fonction f de ces n entrées. Cependant, en général, la fonction d'évaluation, ou fonction d'activation, f ne prend qu'un seul argument. Nous introduisons donc une autre fonction g réduisant les n entrées en une seule. De plus, afin de représenter les différentes intensités des signaux perçus par un neurone, des poids sont associés aux entrées de ce neurone. Ces deux nouvelles caractéristiques sont représentées à la figure 1.3 (b).

(a)



(b)

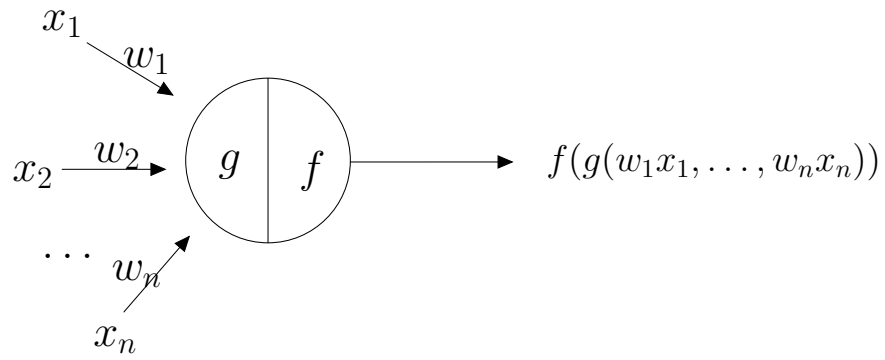


FIGURE 1.3: (a)Représentation générale d'un neurone, (b) Représentation d'un neurone où les n entrées sont réduites à une seule grâce à la fonction g.

Dans la section sur les neurones biologiques, nous avons également vu qu'un neurone pouvait déclenché un potentiel d'action lorsque les différents influx se propageant le long de l'axone atteignent un certain seuil. Il est appelé seuil d'activation et peut être modéliser à l'aide d'une somme pondérée. Cette somme représente ainsi un choix possible pour la fonction g. Ainsi, nous dirons qu'un neurone artificiel sera activé lorsque la relation

$$\sum_{i=1}^{n} w_i x_i \geq b \quad \text{ou encore} \quad \left(\sum_{i=1}^{n} w_i x_i \right) - b \geq 0$$

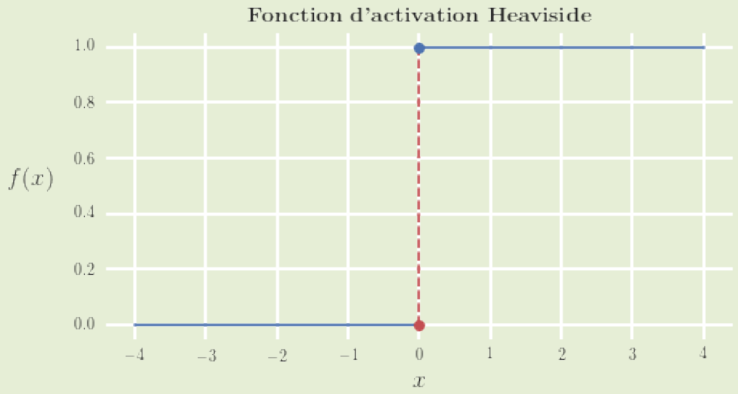
est vérifiée et où b représente le seuil d'activation. Lorsqu'un neurone est activé, nous dirons que sa sortie vaut 1 et dans le cas contraire, elle prendra la valeur 0. Cette sortie correspond donc à la fonction d'activation et est appelée fonction de Heaviside. Il existe bien d'autres exemples de fonction d'activation. Outre la fonction de Heaviside, une autre bien connue est la fonction sigmoïde.

Fonction de Heaviside

Equation :

$$\forall\, x \in \mathbb{R},\ f(x) = \begin{cases} 1 & \text{si } x \geq 0, \\ 0 & \text{sinon.} \end{cases}$$

Graphe :

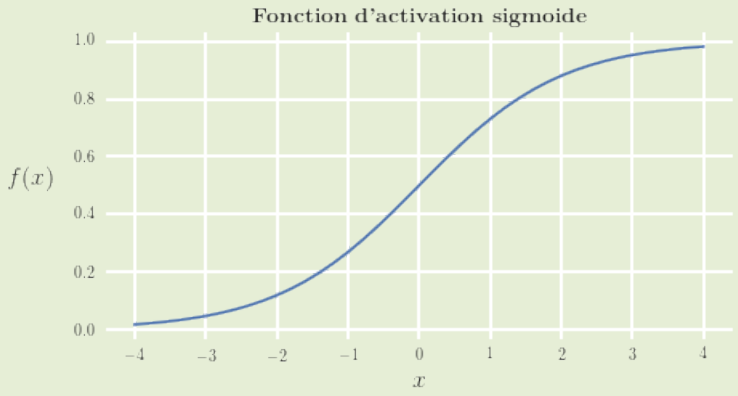


Fonction sigmoïde

Equation :

$$\forall\, x \in \mathbb{R},\ f(x) = \frac{1}{1 + \mathrm{e}^{-x}}$$

Graphe :



1.3 Réseau de neurones artificiels

La modélisation de fonction à l'aide de neurones artificiels peut différer d'une fonction à l'autre. En effet, certaines ne nécessitent qu'un seul neurone tandis que d'autres en requièrent plusieurs formant ainsi un réseau. Ces réseaux sont constitués de plusieurs couches, une couche d'entrée, une ou plusieurs couches intermédiaires, appelées couches cachées et une couche de sortie.

À titre d'exemple, tiré du mémoire de Piazza [13] et de l'article de Rojas [5], les fonctions logiques AND et OR peuvent être modélisées par un seul neurone, cependant ce n'est pas le cas de la fonction XOR. En 2 dimensions, ces fonctions prennent en entrée deux valeurs (x_1, x_2), où x_i peut valoir soit faux, dénoté par la valeur 0, soit vrai, dénoté par 1. De ce fait, le résultat de la fonction vaudra alors soit 0 (faux), soit 1 (vrai). Ces résultats et entrées sont représentés à la table 1.1 (1) pour chaque fonction. Sur base de cette table, nous pouvons représenter géométriquement ces trois fonctions. Pour ce faire, nous traçons dans un graphe les 4 points (x_1, x_2) de la table. Ces points sont de couleur rouge si la sortie correspondante vaut 0 et de couleur verte dans le cas contraire, comme illustré à la figure 1.4. De cette figure, nous pouvons remarquer que pour les deux premières fonctions, AND et OR, les points verts peuvent être séparés par une droite des points rouges. Cette droite a pour équation, $w_1x_1 + w_2x_2 = b$ où b correspond au seuil d'activation du neurone. Nous avons donc bien que le neurone est activé si $g(w_1x_1, w_2x_2) \geq b$ et que sa sortie $f(g(w_1x_1, w_2x_2))$ vaut 1 dans ce cas. Par contre, en ce qui concerne la fonction XOR, il est impossible de séparer ces points avec une seule droite. Cela vient du fait qu'il y a une contradiction dans les conditions d'activation du neurone, représentées dans la table 1.1 (2). En effet, si nous prenons les deuxième et troisième conditions de cette table, nous avons que $w_2 \geq b$ et $w_1 \geq b$ et de ce fait $w_1 + w_2 \geq b$. Cette dernière inégalité contredit la dernière condition dans la table, à savoir $w_1 + w_2 < b$. La fonction XOR ne peut donc pas être modélisée avec un seul neurone.

Cette notion est la séparabilité linéaire. Deux ensembles X et Y dans un espace à n dimensions sont linéairement séparables s'il existe des réels $w_1, \ldots, w_{n+1}$ tels que pour tout $(x_1, \ldots, x_n) \in X$ et pour tout $(y_1, \ldots, y_n) \in Y$,

$$\sum_{i=1}^{n} w_i x_i \geq w_{n+1} \text{ et } \sum_{i=1}^{n} w_i y_i < w_{n+1}$$

Cela permet également de définir la notion de fonction linéairement séparable. Une fonction sera linéairement séparable si ses entrées peuvent former des ensembles linéairement séparables. De ce fait, si nous appliquons cette notion aux réseaux de neurones artificiels, un réseau constitué uniquement d'un neurone ne peut modéliser que des fonctions linéairement séparables.

(1)

x_1	x_2	AND	OR	XOR
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

(2)

x_1	x_2	XOR	Condition sur g
0	0	0	$g = 0 < b$
0	1	1	$g = w_2 \geq b$
1	0	1	$g = w_1 \geq b$
1	1	0	$g = w_1 + w_2 < b$

TABLE 1.1: Table de vérité des fonctions logiques AND, OR et XOR (1). Ainsi que le tableau représentant les sorties attendues de la fonction logique XOR et les conditions sur $g = g(w_1x_1, w_2x_2)$ pour que ces sorties soient respectées selon le seuil d'activation b (2).

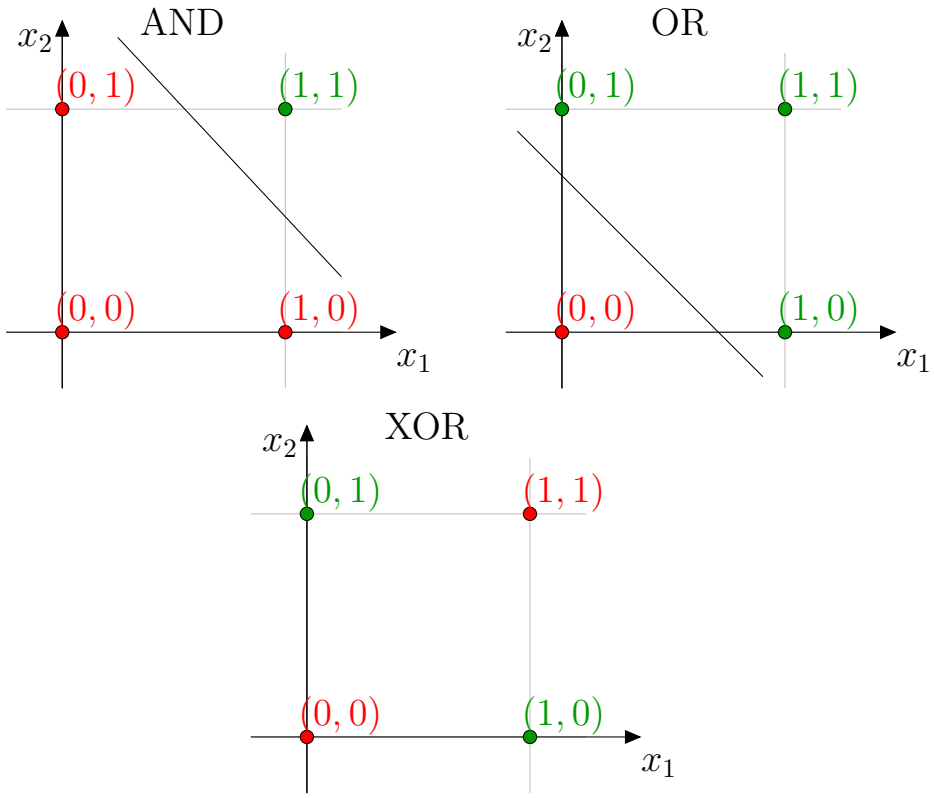


FIGURE 1.4: Représentations géométriques des fonctions logiques AND, OR et XOR.

Une autre caractéristique des réseaux de neurones réside dans la manière dont l'information est propagée. En effet, certains réseaux sont dits feedforward tandis que d'autres sont appelés récurrents. Dans le premier, le flux va strictement des neurones en entrée vers les neurones de sortie sans revenir en arrière, contrairement au second dans lequel il existe des connexions de retour, comme illustré à la figure 1.5.

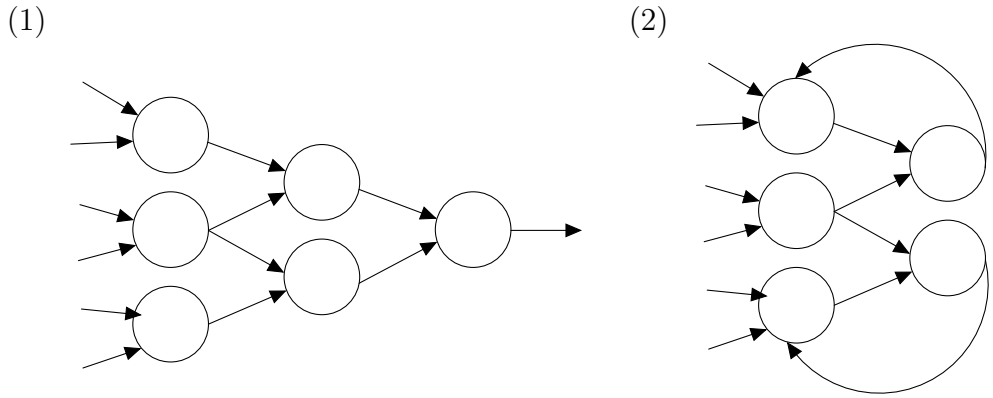


FIGURE 1.5: Exemples de réseaux feedforward (1) et récurrent (2).

1.4 Modularité

Comme dit précédemment dans la section 1.1, chez l'être humain, un réseau de neurones évolue dans le temps. En effet, nous avions dit que lorsque un stimulus est répété régulièrement, cela va maintenir la connexion entre un groupe de neurones, tandis qu'elle va tendre à disparaître dans le cas contraire. Afin de modéliser cette notion de groupe de neurones, nous allons introduire la notion de modularité.

La notion de modularité traduit une structure divisée en communauté dans un réseau ou un graphe. Ainsi, les nœuds d'un réseau, c'est-à-dire les neurones dans notre cas, sont divisés en groupes dans lesquels les connexions sont très nombreuses au sein de ceux-ci mais pauvres entre eux. Si nous prenons un réseau quelconque (hors du contexte des réseaux de neurones), nous pouvons illustré cette structure telle qu'à la figure 1.6. Dans ce graphe, le réseau est divisé en trois communautés.

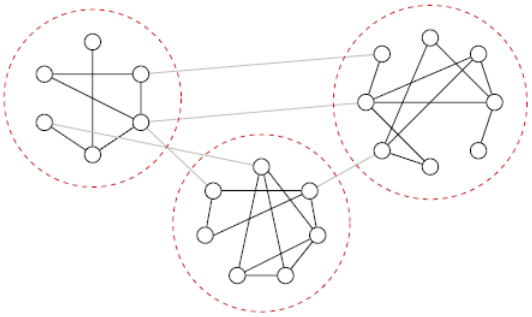


FIGURE 1.6: Exemple de graphe divisé en communauté. Source : [7]

L'étude de cette structure modulaire dans les réseaux est liée à celle du partitionnement en théorie des graphes. Il consiste à diviser un graphe en partitions tout en minimisant certains critères. Dans notre cas, cela reviendrait à minimiser le nombre de connexions entre les différentes communautés. Malheureusement, dans la plupart des cas, le partitionnement exact de graphe est un problème NP-complet. Cependant, différents algorithmes heuristiques ont été développés et donnent des résultats satisfaisants.

Une de ces méthodes est celle de Newman et Girvan ([7], [8], [9], [10]) et elle est basée sur la mesure de centralité des connexions. Cette mesure peut être définie de différentes manières. Cependant, elle doit avoir la particularité d'être une mesure qui favorise les connexions entre les communautés et au contraire défavorise celles intra-communautaire. Un moyen de la calculer est la méthode de Freeman [11]. La mesure de centralité d'un sommet i est alors donnée par le nombre de plus courts chemins entre deux autres sommets passants par ce sommet i. Généralement, l'algorithme de plus courts chemins utilisé est celui appelé parcours en largeurs (ou encore Breadt First Search ou BFS en anglais) :

1. Assigner au sommet initial j une distance $d_j = 0$ ainsi qu'un poids $w_j = 1$.
2. A chaque sommet k adjacent à j, assigner $d_k = d_j + 1$ et $w_k = w_j = 1$.
3. Pour chaque sommet h adjacent à un des sommets k, effectuer une des trois possibilités suivantes :

   - ⋆ Si aucune distance n'a été assignée au sommet h, mettre $d_h = d_k + 1$ et $w_h = w_k$.
   - ⋆ Sinon si $d_h = d_k + 1$, augmenter le poids de w_k, c'est-à-dire $w_h = w_h + w_k$.
   - ⋆ Sinon si $d_h < d_k + 1$, ne rien faire.

4. Répéter à partir de l'étape 3 jusqu'à ce qu'il n'y ait plus de sommet sans distance.

Pour revenir à la mesure de centralité, celle-ci peut-être donnée par l'algorithme suivant :

1. Calculer les plus courts chemins jusqu'au sommet i à partir de chacun des autres sommets k.
2. Assigner $b_{ik} = 1$ à chacun des sommets.
3. Parcourir chacun des sommets k, du plus éloigné au plus proche du sommet i et ajouter b_{ik} à b_{ih}, où h est le sommet précédent k. Dans ce cas,
   $b_{ih} = b_{ih} + b_{ik}$. Si le sommet k possède plus d'un prédécesseur, b_{ik} est divisé de manière égale entre chacun d'eux.
4. Une fois l'étape 3 terminée b_{ik} représente le nombre de plus courts chemin passant par k jusqu'au sommet i. Il faut ensuite répéter le processus à partir de l'étape 1 pour chacun des sommets du réseaux.

Si n est le nombre de neurones dans le graphe, la mesure de centralité du sommet j est alors donnée par

$$\sum_{i=1}^{n} b_{ij}.$$

Cette somme représente le nombre de plus court chemin passant par le sommet j. Plus sa valeur est élevée, plus cela signifie que ce sommet est un sommet intermédiaire et fait donc partie d'une connexion entre deux modules.

L'algorithme de Newman et Girvan permettant de trouver une partition du réseau est ainsi donné par ces quatre étapes :

1. Calculer une mesure de centralité pour toutes les connexions du réseau.
2. Identifier celle avec la mesure la plus élevée et la supprimer du réseau.
3. Recalculer cette mesure pour toutes les connexions restantes
4. Recommencer à l'étape 2.

Maintenant que nous avons cet algorithme, une question se pose : comment savoir si les communautés qui ont été identifiées par l'algorithme sont bonnes ? Pour y répondre, nous allons définir une mesure qui quantifie la qualité d'une division particulière du réseau.

Pour ce faire, Newman et Girvan considèrent une division du réseau en k communautés. Ensuite, ils construisent une matrice E symétrique et de dimension $k \times k$. L'élément e_{ij} de celle-ci est défini comme le rapport de toutes les connexions du réseau qui relient les nœuds de la communauté i à la communauté j. De ce fait, les éléments diagonaux correspondent au rapport des côtés qui connectent les sommets dans une même communauté.

Si la partition du réseau est bonne, la trace de cette matrice sera élevée. Cependant, cette dernière n'est pas à elle seule un bon indicateur. En effet, si nous considérons, par exemple, tous les nœuds dans une seule communauté, la trace vaudra 1, valeur la plus élevée qu'elle peut prendre, mais elle ne nous donne aucune information sur la structure du réseau. Pour remédier à cela, les auteurs ont introduit l'élément $a_i = \sum_j e_{ij}$, c'est-à-dire le rapport des connexions dont une des extrémités est liée à un nœuds de la communauté i. Dans un réseau avec les mêmes communautés mais où les connexions entre les sommets sont aléatoires, nous avons que $e_{ij} = a_i a_j$. Dès lors, la mesure de modularité est définie comme

$$Q = \sum_i (e_{ii} - a_i^2).$$

De manière équivalente, en posant k le nombre de modules dans le réseau, L le nombre de connexions, l_i le nombre de connexions dans le module i et d_i la somme des degrés des nœuds dans le module i, nous pouvons également définir Q comme suit :

$$Q = \sum_{i=1}^{k} \left[\frac{l_i}{L} - \left(\frac{d_i}{2L} \right)^2 \right]. \tag{1}$$

Cette mesure peut prendre des valeurs comprises dans l'intervalle $[-1, 1]$. Si le nombre de côtés dans les communautés correspond aux communautés aléatoires alors elle va tendre vers 0. Elle sera positive si le nombre de connexions est meilleur que ce qu'on pourrait atteindre aléatoirement et la valeur maximale, $Q = 1$, indique un réseau dont la structure est fortement communautaire.

Voyons maintenant une autre métode permettant également d'identifier des communautés, la méthode de Louvain [12]. Cet algorithme est divisé en deux phases qui sont ensuite répétées. Initialement, une communauté est affectée à chacun des noeuds du réseau. Il y a donc autant de communautés que de sommets. Ensuite, pour chaque noeuds i, il faut considérer les noeuds voisins j et évaluer le gain de modularité lorsque le sommet i est placé dans la communauté du noeud voisin j. Le sommet i sera ensuite placé dans la communauté correspondant au gain maximal si celui-ci est positif. Dans le cas contraire, il restera dans sa communauté. Ce processus est ensuite itéré sur chacun des noeuds jusqu'à ce que plus aucune amélioration ne soit possible. Cela constitue la première phase de l'algorithme. En ce qui concerne la seconde phase, elle consiste à créer un nouveau réseau. Dans celui-ci, les communautés du réseau initial sont considérées comme les noeuds du nouveau. La première phase peut ensuite être appliquée à ce nouveau réseau.

Avant d'appliquer ces méthodes à des exemples, nous allons définir la matrice d'adjacence d'un graphe. Supposons dans un premier temps que les connexions entre les sommets n'aient pas de poids et ne soient pas orientées, c'est-à-dire que lorsqu'il y a une connexion entre le sommet i et j, elle est aussi bien dirigée de i vers j qu'inversément. Ainsi, lorsqu'il existe une connexion entre le sommet i et j, un 1 est inscrit à l'intersection de la ligne i et la colonne j (et inversément) de la matrice d'adjacence, comme illustré à la figure 1.7. Dans le cas contraire, nous mettrons 0. Cette matrice est donc carrée et symétrique. Par contre, si nous considérons maintentant un graphe orienté, la matrice ne sera plus symétrique. Si la connexion va du sommet i vers j, le 1 ne sera inscrit qu'à l'intersection de la ligne i et la colonne j. De plus, si les connexions ont des poids w_i, nous remplacerons les 1 par les poids w_i.

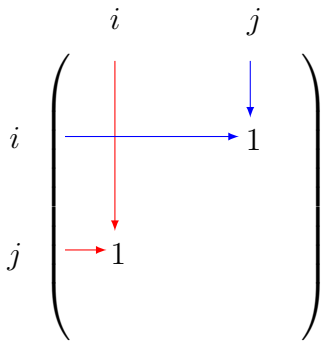


FIGURE 1.7: Explication de la matrice d'adjacence d'un réseau.

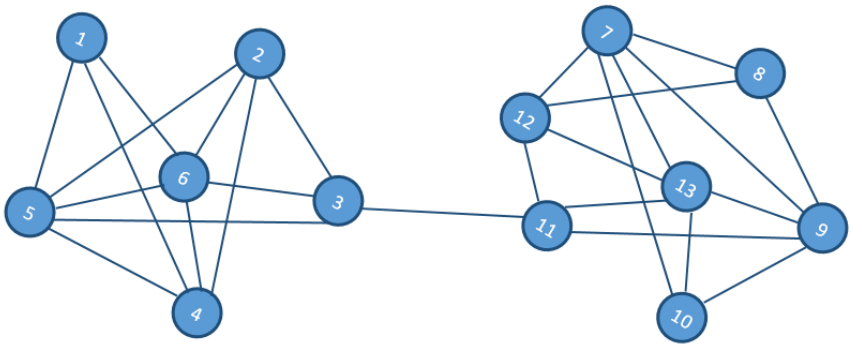
Considérons maintenant les deux exemples à la figure 1.8. Leurs matrices d'adjacence sont données à la figure 1.9. Visuellement, nous pouvons identifier deux communautés dans le premier cas tandis que le second semble formé un seul module. De ce fait, selon la définition de la modularité à l'équation 1, $Q = 0$. Nous pouvons également calculer ce que vaut Q dans le premier exemple. Pour ce faire, nous considérons la partitionnement dont la première communauté est composée des noeuds 1 à 6 et la seconde des noeuds restants. Le nombre total de connexions dans le réseau est de 27 tandis qu'il y en a respectivement 12 et 14 dans chacun des modules. De plus, le somme des degrés des sommets de la première communauté est de 25 et de 29 pour la seconde, ce qui nous donne

$$Q = \frac{12}{27} - \left(\frac{25}{54}\right)^2 + \frac{14}{27} - \left(\frac{29}{54}\right)^2 \approx 0.4602.$$

Vérifions maintenant si nous obtenons le même résultats avec les deux méthodes expliquées précédemment. Pour ce faire, nous utilisons le code Matlab développé par le M.I.T pour la méthode de Newman et Girvan (annexe A) et par Mika Rubinov (annexe B) pour la métode de Louvain. Tout comme nous l'avons calculé, ces deux méthodes nous donnent une modularité d'approximativement 0.4602 et identifient le même partitionnement que celui que nous avons considéré pour l'exemple 1. En ce qui concerne l'exemple 2, les deux méthodes distinguent deux communautés, une composée des noeuds 1, 2 et 5 et l'autre des noeuds restants. Cependant la modularité d'un tel partitionnement n'est que d'environ 0.04545.

Pour chacun des exemples, nous allons maintenant calculer une mesure Q de référence. Pour ce faire, nous prenons 1000 réseaux aléatoires dont le nombre de noeuds et de connexions correspondent au réseau initial. Nous calculons ensuite la mesure Q de ces réseaux et nous définissons le Q de référence comme la moyenne de toutes ces valeurs. Pour le premier exemple, nous obtenons $Q_{ref} = 0.216 \pm 0.033$ et pour le second $Q_{ref} = 0.035 \pm 0.018$. Dans le second cas, nous pouvons voir que la mesure de modularité n'est pas meilleur que celle de référence, nous pouvons dire que cet exemple ne possède pas une structure modulaire. Par contre, en ce qui concerne le premier exemple, nous avons une mesure de modularité meilleure que celle de référence, ainsi ce réseau possèdent des communautés.

(1)



(2)

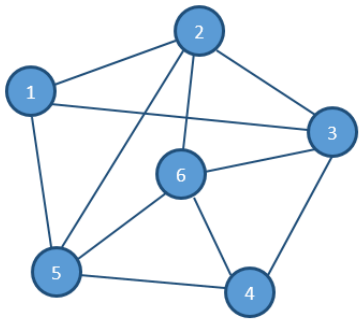


FIGURE 1.8: Exemples de réseaux avec (1) et sans communautés (2).

$$\begin{pmatrix}
0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1
\end{pmatrix} \text{ et } \begin{pmatrix}
0 & 1 & 1 & 0 & 1 & 0 \\
1 & 0 & 1 & 0 & 1 & 1 \\
1 & 1 & 0 & 1 & 0 & 1 \\
0 & 0 & 1 & 0 & 1 & 1 \\
1 & 1 & 0 & 1 & 0 & 1 \\
0 & 1 & 1 & 1 & 1 & 0
\end{pmatrix}.$$

FIGURE 1.9: Matrices d'adjacence respectivement de l'exemple 1 et 2.

1.5 Conclusion

Dans ce chapitre, nous avons vu différentes caractéristiques d'un neurone biologique. Afin de construire nos neurones artificiels, nous en avons gardé quelques-unes dont des éléments de leur structure, à savoir les dendrites, le corps cellulaire, l'axone et les synapses. Nous avons également modélisé le fait que, pour qu'un neurone soit activé, son excitation doit dépasser un certain seuil. Nous avons également défini différents types de réseaux, tels que les réseaux constitués d'un seul neurone et les multi-couches, les réseaux feedforward et les récurrents. Nous avons ensuite introduit la notion de modularité et introduit deux algorithmes permettant d'identifier des partitionnements dans des graphes, ainsi qu'une mesure dont l'objectif est de quantifier la modularité de ces partitionnements.

Chapitre 2

Algorithmes génétiques

Dans le chapitre précédent, nous avons vu qu'un réseau de neurones biologiques pouvait évoluer dans le temps grâce à l'apprentissage. Il existe plusieurs méthodes permettant de modéliser ce phénomène. Une d'entre elles est une méthode d'optimisation, appelée algorithme génétique. L'objectif de ce chapitre, est introduire ce concept. Pour ce faire, nous nous baserons sur le cours donné par Pr. Pierre Collet [14] pour décrire la méthode et sur le mémoire de Salvino Piazza [13] pour l'appliquer au réseaux de neurones.

2.1 Concepts de base

2.1.1 Mise en place du vocabulaire

Tout comme les réseaux de neurones, les algorithmes génétiques s'inspirent d'un principe biologique. Le monde, tel que nous le connaissons aujourd'hui, est le résultat d'un long méchanisme d'évolution des espèces vivantes. Dans ce méchanisme intervient le principe de la théorie de l'évolution de Darwin. Cela consiste en la sélection des individus les mieux adaptés à leur environnement, appelée sélection naturelle. De plus, interviennent également dans ce mécanisme de l'évolution, la reproduction des espèces et leurs mutations. Les algorithmes génétiques ont été construits sur ce même schéma. Les différents éléments constituant ces algorithmes sont représentés par des termes provenant de la biologie. Nous allons définir ces éléments sur base du mémoire de Romain Hendrickx [15].

Gène :

En biologie, les gènes sont des séquences d'ADN. Ce sont les informations contenues dans ces séquences qui nous caractérisent. Ainsi, c'est un certain nombre de gènes qui contribue, par exemple, à la couleur de nos yeux.

Allèle :

"Se dit d'une variante d'un gène, résultant d'une mutation et héréditaire, assurant la même fonction que le gêne initial mais selon ses modalités propres" (Petit Larousse [16]). Par exemple, si nous reprenons le gène qui contribue à la couleur de nos yeux, les différents allèles de ce gènes sont les différentes couleurs que peuvent avoir nos yeux.

Chromosome :

Les chromosomes sont de longs filaments d'ADN fortement condensés et qui sont composés de gènes. Il peuvent être modélisé tel qu'à la figure 2.1

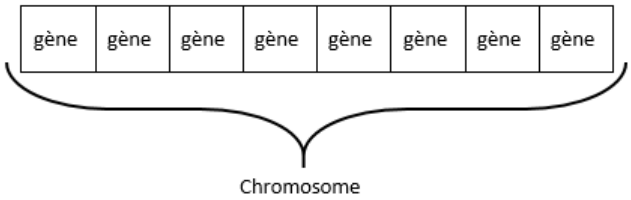


FIGURE 2.1: Représentation schématique d'un chromosome.

Individu et population :

Un individu est constitué d'un ou plusieurs chromosomes tandis qu'un ensemble d'individus forme une population.

Génotype :

"Ensemble des gènes d'un individu, du point de vue des caractéristiques de leurs allèles", définition du Petit Larousse [16].

Phénotype :

"Ensemble des caractères apparents (morphologiques, chimiques, ...) d'un organisme, d'une cellule, résultant de l'expression du génotype et de l'influence du milieu", définition du Petit Larousse [16].

Fitness :

La fitness est une mesure de l'adaptabilité d'un individu à son environnement et détermine sa chance de procréer selon l'hypothèse de Darwin.

2.1.2 Méthode

Comme introduit en début de chapitre, les algorithmes génétiques sont des méthodes d'optimisation. Pareillement à toutes ces méthodes, l'objectif est de trouver une solution optimale minimisant une fonction objectif f, qui correspond à la fonction de fitness. Nous verrons dans le chapitre suivant en quoi cette méthode nous intéressent et à quoi se rapporte la fitness dans notre problème.

Quelque soit le problème à optimiser, il est important de mettre en place l'algorithme. Pour ce faire, il faut définir une représentation de la solution du problème (par exemple, représenter la hauteur d'un mât à l'aide d'un vecteur de réels) et une manière d'évaluer cette solution, c'est-à-dire définir la fonction de fitness. Supposons ces éléments posés, nous pouvons maintenant décrire les différentes étapes de l'algorithme tel qu'il est illustré à la figure 2.2.

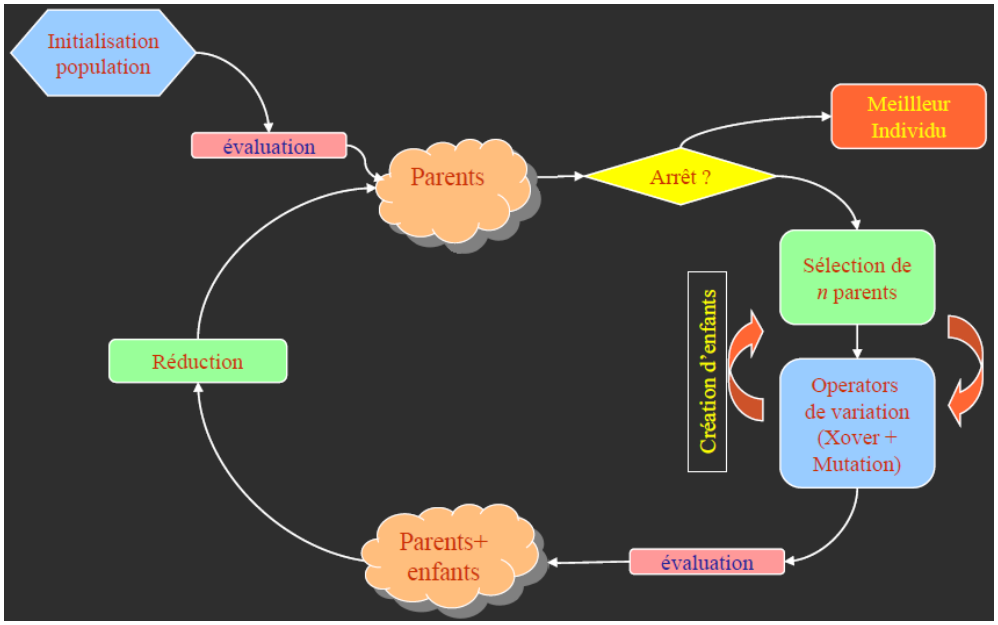


FIGURE 2.2: Schéma général des algorithmes génétiques. Source :[14]

1. **Initialisation de la population**

   Le principe de l'algorithme consiste a sélectionner les individus les mieux adapter. Il est donc nécessaire de créer une population d'individus à l'aide de la même représentation que celle choisie pour la solution. Afin de préserver la diversité, ces individus sont générés de manière aléatoire. Il est, en effet, important de la préserver car elle nous permet d'obtenir des solutions auxquelles nous n'aurions pas forcément pensé. Une fois la population initialisée, chacun des individus est ensuite évalué à l'aide de la fonction de fitness.

2. **Vérification des critères d'arrêts**

   Nous vérifions si un des individus correspond aux critères d'arrêts. Ces critères sont par exemple un nombre maximal d'itérations, l'algorithme qui ne parvient pas à trouver une meilleure solution après un nombre déterminé d'itération, ... Si nous trouvons un tel individu, l'algorithme est alors arrêté et notre solution est le meilleur individu qui correspond à ces critères.

3. **Création d'individus enfants**

   Si l'algorithme n'est pas arrêté, nous créons des enfants. Pour chaque enfant à créer, nous sélectionnons n parents auxquels nous appliquons des opérations de variation, que nous développerons par la suite. Nous évaluons ensuite les nouveaux individus et les ajoutons à notre population.

4. **Réduction de la population**

   Dans cette phase, nous réduisons notre population avant de recommencer le processus à l'étape 2. Cette étape est importante. En effet, si nous ne passons pas par cette phase, notre population va croître de plus en plus. Cela aurait pour conséquence négative d'augmenter le temps de calcul de la solution, voir même de le faire exploser.

Dans cette méthode, nous avons parlé de sélection de n parents, de réduction de la population et d'opérations de sélection. Nous allons maintenant approfondir ces différentes notions.

Sélection

Le terme "sélection" englobe aussi bien la sélection des n parents que la réduction. En effet, cette réduction consiste à sélectionner un certain nombre d'individus de notre population parents et enfants afin d'en faire une nouvelle population de parents. Une des différences entre ces deux sélections est que pour la création d'un individu, un parent peut être sélectionné à plusieurs repises (nous parlons ainsi de sélection avec remise) tandis que pour la réduction, un individus ne pourra être tiré qu'une et une seule fois.

Comme tout algorithme d'optimisation, les algorithmes génétiques peuvent converger de manière prématurée et, de ce fait, tomber dans un minimum locale. Afin de lutter contre ce phénomène, il est important d'afficher la courbe de convergence du meilleur individu, tel qu'illustré à la figure 2.3.

Supposons que nous disposions de 2h de temps de calcul. Si nous regardons la courbe bleue, nous pouvons remarquer qu'elle décroit trop lentement. En effet, nous voyons qu'au bout de 2h, elle est toujours occupée à décroitre et que nous n'avons pas de convergence. Dans ce cas, nous parlons de pression de sélection trop faible. Il faudrait donc un opérateur de sélection plus fort qui permette à l'algorithme de converger plus rapidement. Mais il faut alors faire attention à ne pas tomber dans le cas opposé et converger trop rapidement, comme illustré par la courbe jaune. Dans ce cas, nous perdons du temps de calcul car durant un grand laps de temps, la solution n'a pas ou trop peu évolué. Nous parlons de pression de sélection trop forte. Par contre, la courbe verte représente une bonne convergence. La pression de sélection est bien maitrisée.

Mais il est important de noter que cela dépend fortement du temps de calcul dont nous disposons. En effet, supposons maintenant que nous possèdions 15 minutes de temps de calcul. La courbe bleue présente toujours une pression de sélection trop faible tandis que la courbe verte correspond maintenant à une pression de sélection trop forte.

Revenons maintenant à la sélection des individus. Il existe différentes méthodes de sélection. Nous allons en décrire quelques unes, à savoir la sélection par roulette, le tournoi et l'élitisme.

1. **Roulette**

   Cet opérateur est l'opérateur historique des algorithmes génétiques. Il se base sur le principe du jeu de la roulette au casino. Dans ce jeu, la roulette est divisée en case de même proportion. Ainsi lorsque nous faisons tourner une bille dans cette roulette, la probabilité qu'elle a de tomber dans une case est la même pour toutes. Le principe est le même pour cet opérateur mais avec des proportions différentes comme illustré à la figure 2.4.

   Chaque case correspond à un individu de la population et la probabilité p_i de sélectionner une case i, est proportionnelle à sa fitness. Elle est donnée par l'équation 1 ci-dessous.

$$p_i = \frac{f(x_1)}{\sum_{i=1}^{m} f(x_i)} \tag{1}$$

   où x_i correspond au i-ème individu d'une population de taille m.

Ce processus assez simple comporte toutefois des inconvénients. En effet, il dépend de la variance de la fitness. Ainsi si la variance est trop petite cela reviendrait à sélectionner les individus de manière pratiquement aléatoire. À l'opposé, avec une variance trop grande, nous sélectionnerions de manière trop importante les meilleurs individus et cela provoquerait une perte de la diversité de la population. Un autre inconvénient de cette méthode est qu'elle est très couteuse car elle demande de connaitre les évaluations de tous les individus de la population.

2. **Tournoi**

Il existe différents types de sélection par tournoi. Le premier consiste en un tournoi déterministe. Son principe est de choisir n individus de manière uniforme et de prendre ensuite le meilleur. Ainsi si nous prenons n égal à la taille de la population, cela revient à prendre le meilleur individu de la population et correspond à une sélection complètement déterministe. Au contraire, si n vaut 1, on a une sélection complètement aléatoire. En pratique, il est conseillé de prendre n entre 2 et 7. Augmenter, le nombre d'individus tirés permet d'augmenter la pression de sélection et ainsi de faire converger l'algorithme plus rapidement. Au contraire, si cette pression est trop grande et que notre algorithme converge trop rapidement, il suffit de diminuer n. L'avantage de cette méthode est qu'elle possède un coût très faible. En effet, elle ne demande pas de connaitre tous les individus de la population. Nous sélectionnons uniquement n individus de manière aléatoire et il n'est pas nécessaire de connaitre leur évaluation pour les sélectionner, seulement pour trouver ensuite le meilleur. Cependant cela entraine également un inconvénient, nous risquons d'évaluer plusieurs fois le même individu et d'ainsi perdre du temps. Un autre inconvénient est que nous ne pourrons jamais sélectionner les $n - 1$ pires individus de la population.

Dans le cas d'un tournoi entre 2 individus où la pression de sélection est trop forte, il faudrait pouvoir diminuer cette sélection. Mais prendre n égal à 1 n'est pas une solution, car cela revient à une sélection complètement aléatoire. Ainsi l'idée est de réaliser un tournoi non plus déterministe mais stochastique. Celui-ci consiste à choisir uniformément 2 individus et à prendre le meilleur avec une probabilité t entre 0.5 et 1. Ainsi, si t vaut 0.5, cela revient à une sélection aléatoire tandis que si t vaut 1, cela correspond simplement à un tournoi déterministe entre 2 individus. Si t vaut par exemple 0.7, nous avons la possiblité de choisir l'individu qui n'est pas le meilleur des deux individus sélectionnés. Cela correspond à réduire la pression de sélection par rapport à un tournoi déterministe binaire.

3. **Elitisme**

Il faut savoir que l'élitisme n'est pas une sélection à proprement parler. Il faut toujours la combiner avec un opérateur de sélection. La notion d'élitisme consiste à sélectionner les x meilleurs individus de la population et à les garder tel quel dans la population d'enfants. Lorsque $x = 1$, cela est appelé élitisme fort. Pour fabriquer les autres enfants, les parents sont alors sélectionnés avec une autre méthode comme celles vues précédemment.

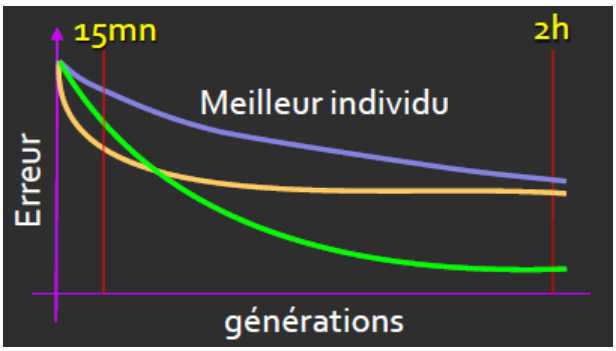


FIGURE 2.3: Exemples de courbe de convergence du meilleur individu. Source :[14]

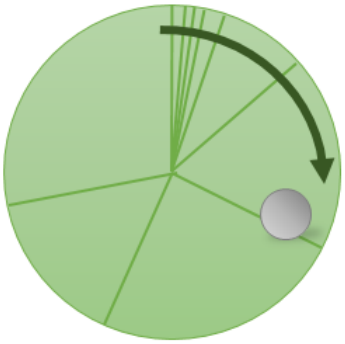


FIGURE 2.4: Représentation de la sélection par roulette.

Une fois les parents sélectionnés, nous créons de nouveaux individus, appelés enfants, à l'aide d'opérateurs de variation. Nous allons donc voir plus précisément en quoi consiste ces opérateurs.

Opérateurs de variation

Les opérateurs de variation permettent de produire de nouveaux individus à l'aide d'individus parents. Cela consiste à modifier les gènes parents afin de créer des enfants. Nous allons voir en particulier, les opérations de croisement, aussi appelé crossover, et de mutation.

1. **Crossover**

   Le crossover consiste à mélanger les gènes des parents. Pour ce faire, il existe différentes méthodes tel que le crossover monopoint, multipoint et uniforme.

   Comme son nom l'indique le crossover monopoint sélectionne de manière aléatoire un point du chromosome. Il échange ensuite la fin des chromosomes, tel qu'illustré à la figure 2.5 (1). De manière similaire au monopoint, le crossover multipoint échange plusieurs parties des chromosomes, comme nous pouvons le voir à la figure 2.5 (2). Dans le cas du croisement uniforme, pour chaque gène, nous choisissons aléatoirement le gène correspondant chez le premier ou le second parent. Mais il faut faire attention à ne pas séparer des gènes qui sont très corrélés. Il faut également prendre en compte que l'efficacité du croisement diminue au fil des générations. En effet, les individus vont de plus en plus se ressembler, et échanger des gènes entres individus qui se ressemble déjà a très peu d'influence.

2. **Mutation**

   Contrairement au crossover, la mutation est un opérateur unaire. Elle modifie le code génétique d'un individu sans utiliser d'autres individus. Tout comme pour l'opérateur précédent, il existe différentes mutations.

   La mutations monopoint va sélectionner un gène aléatoirement et le remplacer par une autre valeur, elle aussi choisie au hasard, comme représenté à la figure 2.6 (1). A la figure 2.6 (2) est représenté la mutation multipoint, elle consiste en la même méthode que celle du monopoint. Elle choisit au hasard non plus un gène mais plusieurs. Il existe également une mutation qui modifie totalement tous les gènes d'un individu en les remplaçant par des valeurs choisies aléatoirement.

   L'impact de la mutation est très faible au début. En effet, la population étant initialisée aléatoirement, modifier quelques gènes ne change pas grand chose. Cependant à la fin du processus, l'impact devient très nuisible. Nos individus étant très bons, il est donc difficile de trouver mieux en modifiant des gènes au hasard.

A chacun de ces opérateurs est associé un taux. En règle générale, le taux de crossover indique la probabilité que deux individus soient croisés. Ainsi s'il y a un croisement, les individus enfants sont composés des gènes de chacun de leurs parents. Dans le cas contraire, ils sont une copie de ces derniers. En ce qui concerne le taux de mutation, il représente la probabilité qu'un gène d'un individu soit muté. Le taux de croisement est généralement assez élevé, c'est-à-dire compris entre environ 0.5 et 0.9, tandis que celui concernant la mutation est faible, entre 0.001 et 0.01. En effet, si celui-ci était trop élevé, l'algorithme tendrait vers une recherche aléatoire de la solution.

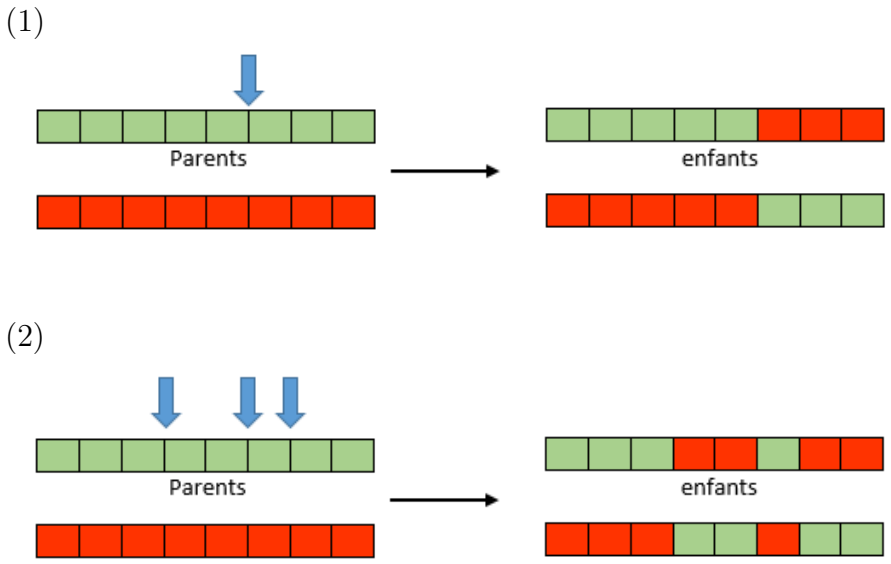


FIGURE 2.5: Opération de crossover monopoint (1) et multipoint (2).

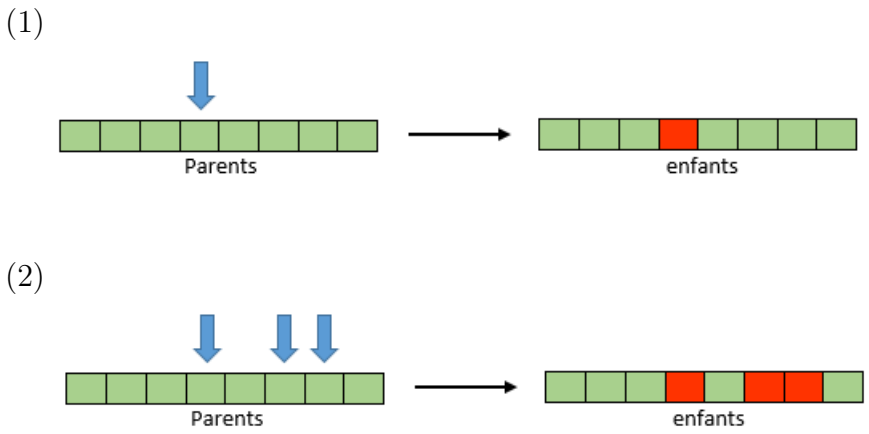


FIGURE 2.6: Opération de mutation monopoint (1) et multipoint (2).

2.2 Application aux réseaux de neurones

Les algorithmes génétiques appliqués aux réseaux de neurones peuvent servir à atteindre différents buts. Ci-dessous, nous avons une liste non-exhaustive de ces différents objectifs.

- Apprendre les poids d'un réseau pour une architecture fixée.
- Trouver l'architecture optimale d'un réseau, incluant la recherche du nombre de neurones ainsi que les connexions optimales entre les neurones.
- Trouver une bonne règle d'apprentissage. L'apprentissage consiste à assimiler ou améliorer des connaissances. De ce fait, cela revient à dire de trouver une bonne règle afin d'effectuer une tâche.
- Diminuer le nombre d'entrées afin de garder uniquement les plus pertinentes.
- Sélectionner les meilleurs éléments des groupes d'apprentissage.

Afin d'obtenir des opérations de variation efficaces, ces opérations sont effectuées sur le génotype, qui correspond au codage du réseau de neurones, plutôt que sur le phénotype, le réseau lui-même. Il existe plusieurs manières de coder un réseaux de neurones. Nous allons maintenant décrire quelques unes de ce méthodes.

Direct :

Dans le cas de ce codage, le réseau est directement encodé dans le chromosome. La transformation du génotype en phénotype est assez trivial. Un exemple de cet encodage est la matrice des connexions. Pour cet exemple, un crossover choisirait de manière aléatoire une des colonnes de la matrice et échangerait les colonnes correspondantes des individus parents. L'opérateur de mutation, quant à lui, changerait aléatoirement la valeur des éléments dans la matrice.

Paramétrique :

Le réseau est encodé à l'aide de paramètres, ce qui permet de diminuer la longueur des chromosomes dans le cas des réseaux de grandes tailles. Par exemple, les paramètres encodés sont le nombre de neurones, le nombre de couches et la mesure de le centralité de chacun des neurones. Ainsi, ajouter un neurone modifie un paramètre plutôt que d'augmenter la taille du chromosome.

Grammaire :

Le codage de grammaire consiste à encoder diffrentes règles qui permettent de développer le réseau. Dans ce cas, pour passer du génotype au phénotype, cela demande un effort important.

2.3 Conclusion

Dans ce chapitre, nous avons défini le vocabulaire des différents éléments qui constituent les algorithmes génétiques. Nous avons ensuite développé la méthode et nous avons vu des opérateurs de sélection ainsi que ceux de variation, le crossover et la mutation. Nous avons, par la suite, appliqué les algorithmes génétiques aux réseaux de neurones.

Chapitre 3

Application au problème

Dans les chapitres précédents, nous avons vu les différents outils nécessaires à la compréhension de notre problème. Nous allons maintenant les appliquer à celui-ci.

3.1 Description

Comme annoncé précédemment, l'étude porte sur un modèle qui a pour tâche la reconnaissance de schémas. Ceux-ci sont constitués de 8 pixels, formant un rectangle de 4 pixels sur 2, chacun ayant une valeur de 0 ou 1. La moitié gauche correspond au motif perçu par le côté gauche d'une rétine tandis que celle de droite est celui perçu par le côté droit. L'objectif est de reconnaître des objets qui proviennent du côté gauche et du côté droit de la rétine. Pour chaque côté de la rétine, un objet existe si les quatre pixels correspondants coïncident avec un des schémas d'un ensemble prédéfini. À la figure 3.1, nous pouvons voir une représentation du problème. Nous avons donc une rétine composée des 8 pixels et pour chaque moitié de la rétine un ensemble d'objets est défini.

Nous avions également annoncé qu'afin d'expliquer l'origine de la modularité dans les réseaux de neurones, N. Kashtan et U. Alon faisaient évoluer leur environnement dans le temps dans le but d'atteindre deux tâches d'apprentissages. De plus, ces dernières sont des combinaisons différentes de même sous-but. Ainsi, dans notre cas, nous avons deux sous-buts, reconnaître la rétine gauche L et reconnaître la rétine droite R. De ce fait, la première tâche correspond à reconnaitre la rétine gauche et la droite, L et R, et la seconde à reconnaitre l'une ou l'autre, L ou R.

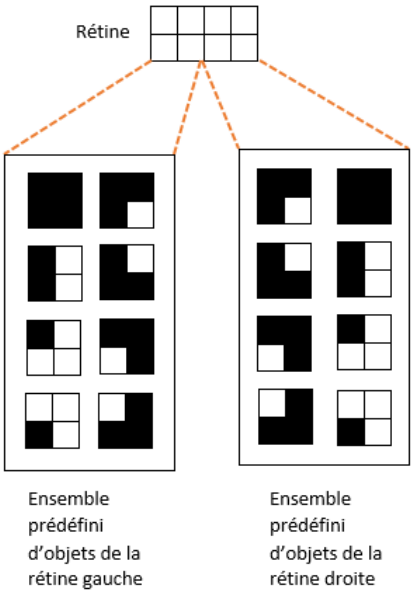


FIGURE 3.1: Représentation du problème.

3.2 Réseau de neurones artificiels

3.2.1 Réseau

En ce qui concerne le réseau de neurones, il est multicouche et de type feed-forward. Il ne contient donc pas de connexions de retour.

Ce réseau prend en entrée un motif provenant d'une rétine. Comme expliqué dans la description du problème, ce motif est constitué d'un total de 8 pixels. Ceux-ci peuvent être considérés comme des neurones sensitifs se connectant à la première couche du réseaux, composée de 8 neurones. Le réseau sera ensuite constitué de deux couches cachées, chacune comprenant respectivement 4 et 2 neurones. Enfin, la couche de sortie comporte, quant à elle, 1 neurone et sa sortie vaut 1 si le motif a été reconnu et 0 dans le cas contraire.

Les connexions entre les neurones se font uniquement entre les neurones appartenant à des couches successives. Dans l'article de référence [1], le nombre d'entrées maximales pour chaque neurone est défini à 3 pour tous les neurones appartenant aux trois premières couches et à deux pour celui de la dernière. Cependant après analyse de la méthode présentée, nous avons décidé de fixer le nombre d'entrées à la valeur maximale. À chaque connexion, un poids de -1 ou 1 est également associé. De plus, si deux entrées proviennent d'un même neurone alors le poids de la connexion vaut la somme des poids de ces deux entrées.

Chaque neurone possède également un seuil d'activation. Il est défini comme une valeur entière appartenant à l'intervalle $[-2^{i-1},\ 2^{i-1}-1]$ où i est le nombre d'entrées maximales pour le neurone considéré. De plus, le neurone est activé si la relation 1 ci-dessous est respectée, où b représente le seuil d'activation, x_i une valeur d'entrée (0 ou 1), w_i le poids de la connexion de cette entrée et n le nombre d'entrées du neurone.

$$\sum_{i}^{n} w_i x_i \geq b. \tag{1}$$

3.2.2 Mesure de la modularité

Afin de quantifier la modularité d'un réseau, nous utilisons la mesure de Newman et Girvan que nous avons définie dans le premier chapitre. Pour rappel, cette mesure est caractérisée par le rapport des connexions dans un module moins la valeur attendue dans un réseau avec les mêmes modules mais des connexions aléatoires dans ceux-ci :

$$Q = \sum_{i=1}^{k} \left[\frac{l_i}{L} - \left(\frac{d_i}{2L} \right)^2 \right],$$

où k est le nombre de modules, L est le nombre de connexions dans le réseau, l_i est le nombre de connexions dans le module i et d_i est la somme des degrés des nœuds dans le module i.

Toutefois, la mesure utilisée dans l'article est une mesure normalisée de Q. Cette nouvelle quantité Q_m est donnée par :

$$Q_m = (Q_{real} - Q_{rand})/(Q_{max} - Q_{rand}).$$

Afin de calculer Q_m, Q_{real} est tout d'abord évalué sur le vrai réseau. Toutefois, ce réseau est modifié en un graphe non-dirigé. Ensuite, Q_{rand} est la valeur moyenne Q de 1000 réseaux aléatoires. Ces réseaux aléatoires sont générés de telle sorte qu'ils préservent la séquence de degré du vrai réseau. En ce qui concerne l'évaluation de Q_{max}, nous utilisons le même algorithme génétique que pour l'apprentissage de nos réseaux. De ce fait ils possèdent les mêmes paramètres (la taille du réseau et les paramètres d'évolution). Seul la mesure de la fitness est différente. Dans ce cas-ci, l'objectif est de maximiser la mesure de la modularité Q. Q_{max} est ensuite estimé comme la moyenne sur 100 simulations du meilleur réseau évolué.

3.3 Algorithmes génétiques

Afin de faire évoluer le réseau de neurones, nous utilisons un algorithme génétique. Mais avant d'entrer plus en détail, définissons la fonction fitness afin de pouvoir évaluer un individu, c'est-à-dire un réseaux de neurones. Celui-ci évolue dans un environnement composés de 100 motifs choisis de manière aléatoire. La fitness est, de ce fait, le pourcentage de motifs dans cet environnement que le réseau a identifié correctement. Cependant, nous avions également parlé de deux tâches d'apprentissage. Ainsi, dans un premier temps nous allons faire évoluer nos réseaux afin qu'il atteignent une objectif fixe, soit L et R, soit L ou R. Dans un second temps, toutes les 20 générations, nous ferons varier la tâche d'apprentissage entre ces deux objectifs. De plus, pour tout individu constitué de plus de 13 neurones, une pénalité de 0.01 est appliquée pour chaque neurone additionnel.

Nous allons maintenant nous attacher à définir la représentation d'un individu. Dans ce problème, un individu correspond donc à un réseau de neurones artificiels. Ce réseau, en génétique, est appelé génome et est constitué de plusieurs gènes. Ces derniers, quant à eux, sont les neurones et sont composés d'un seuil et d'entrées, elles-mêmes caractérisées par l'adresse et le poids de la connexion. Ces divers éléments sont illustrés à la figure 3.4. En ce qui concerne le seuil, le poids d'une connexion et l'adresse d'une entrée, ils sont transformés en objet binaire. Les tables 3.1, 3.2, 3.3 et 3.4 reprennent ces différentes conversions. Un réseau de neurones sera donc un vecteur constitué d'une succession de 0 et de 1.

Maintenant que la fitness et la représentation d'un réseaux sont bien définies, nous allons expliquer pourquoi nous utilisons un algorithme génétique. La fitness d'un réseau que nous avons définie mesure les performances de celui-ci à reconnaître le schéma d'une rétine. Nous ne pouvons donc pas calculer le gradient d'une telle fonction. Il n'est dès lors pas possible d'utiliser des méthodes d'optimisation tel que la descente du gradient. De plus, les algorithmes génétiques sont bien adaptés pour les problèmes dont la représentation de la solution est plus complexe que, par exemple, un simple vecteur de réels.

L'algorithme génétique utilisé est un algorithme standard tel que nous l'avons décrit au chapitre 2. Nous utilisons les opérateurs de variation crossover et mutation combiné à une stratégie élite. La population initiale est générée aléatoirement. Pour chaque génération, l'opérateur de crossover est appliqué entre 2 réseaux choisis avec une probabilité $P_c = 0.5$. De plus, une connexion, un poids ou un seuil d'activation dans chaque réseau de la population est changé avec une probabilité $P_m = 0.5$. En ce qui concerne la stratégie élite, les L meilleurs individus passent inchangé dans la génération suivante remplaçant les L plus mauvais où $L = 150$. De plus, la taille de la population est fixée à $S = 600$.

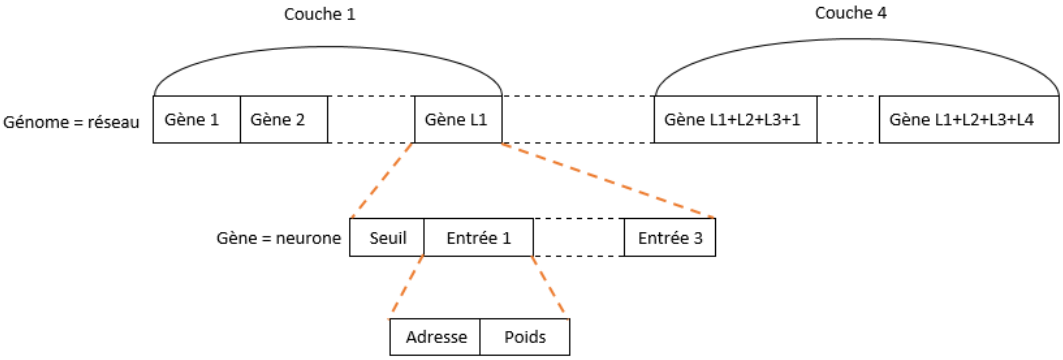


FIGURE 3.2: Représentation d'un réseau de neurones artificiels.

Code du seuil	Seuil du neurone
000	3
001	2
010	0
011	1
100	−2
101	−3
110	−1
111	−4

Code du seuil	Seuil du neurone
00	1
01	0
10	−2
01	−1

TABLE 3.1: Seuil des neurones des trois premières couches dans la table de gauche et de la quatrième couche dans celle de droite.

Code de l'adresse	Adresse physique
000	Rétine - pixel 1
001	Rétine - pixel 2
010	Rétine - pixel 3
011	Rétine - pixel 4
100	Rétine - pixel 5
101	Rétine - pixel 6
110	Rétine - pixel 7
111	Rétine - pixel 8

Code adresse	Adresse physique
000	Neurone 1
001	Neurone 2
010	Neurone 3
011	Neurone 4
100	Neurone 5
101	Neurone 6
110	Neurone 7
111	Neurone 8

TABLE 3.2: Adresses des entrées des neurones de la première couche dans la première table et de la deuxième dans la seconde.

Code adresse	Adresse physique
00	Neurone 9
01	Neurone 10
10	Neurone 11
01	Neurone 12

Code adresse	Adresse physique
0	Neurone 13
1	Neurone 14

TABLE 3.3: Adresses des entrées des neurones de la troisième couche dans la première table et de la dernière dans la seconde.

Code du poids	Poids de la connexion
0	-1
1	1

TABLE 3.4: Poids des connexions.

3.4 Résultats

Pour chacune des tâches d'apprentissage, qu'elle soit fixe ou non, 24 simulations d'évolution ont été réalisées. Lorsque nous faisons évoluer les réseaux de neurones que ce soit pour atteindre une tâche d'apprentissage fixe ou non, après 35 000 itérations (pouvant varier entre 30 000 et 50 000), la solution a une fitness moyenne de 0.91 ($\pm$0.03). Cela signifie qu'elle est capable de reconnaître 91% des schémas de l'environnement. A la figure 3.3, nous pouvons voir l'évolution de la solution en termes de génération lorsque le réseau évolue sous le but fixe L et R, les résultats sont similaire pour l'évolution sous des buts variables.

En ce qui concerne la mesure de la modularité, nous pouvons remarquer dans la table 3.5 qu'il n'y a pas de différence entre les réseaux évolués dans le but d'atteindre une tâche fixe et ceux dont la tâche d'apprentissage est changeante. Nous pouvons également remarquer que le mesure normalisée Q_m est négative. Cela est dû en fait que la valeur moyenne des vrais réseaux de neurones est inférieure à la valeur moyenne des réseaux aléatoires. Nous pouvons donc conclure que dans les deux cas nos réseaux, n'ont pas de structure modulaire. Ces différentes mesures ont été calculées à l'aide de l'implémentation de la méthode de Louvain par Mika Rubinov (annexe B). En effet, cette méthode a été adaptée afin de pouvoir évaluer la mesure de modularité d'un graphe avec des poids négatifs.

Cependant, les résultats observés dans l'article ne sont pas tout à fait les mêmes. En effet, nous pouvons remarquer dans la table 3.6 que les réseaux évoluant sous des buts changeants possèdent une structure plus modulaire que dans l'autre cas. Ces réseaux sont divisés en deux modules, un module permettant de reconnaître le schémas dans la rétine de gauche et un autre pour la rétine droite, le neurone de sortie combine ensuite ces deux modules afin de fournir une réponse. Un tel réseau est illustré à la figure 3.4 (b). Dans le cas où le réseau doit atteindre la tâche fixe L et R, sa structure est tel qu'illustrée à la figure 3.4 (a). En ce qui concerne nos résultats, les réseaux évoluant sous le but fixe L et R possèdent une structure similaire à celle de l'article.

Nous pouvons remarquer que la mesure Q_{rand} dans nos résultats est plus élevées que celle dans l'article. Nous avons donc implémenté une autre manière de l'évaluer. Elle correspond toujours à la valeur moyenne Q de 1000 réseaux aléatoires. Cependant, ceux-ci sont maintenant générés de manière à ce qu'ils préservent le nombre de noeuds, de connexions ainsi que les poids de ces connexions. En procédant de cette façon, nous avons que $Q_{rand} = 0.038 \pm 0.04$ et qu'ainsi $Q_m = 0.07 \pm 0.04$. En ce qui concerne la différence de modularité dans les réseaux évolués sous des buts changeants, cela pourrait être dû à une différence ou encore une erreur d'implémentation.

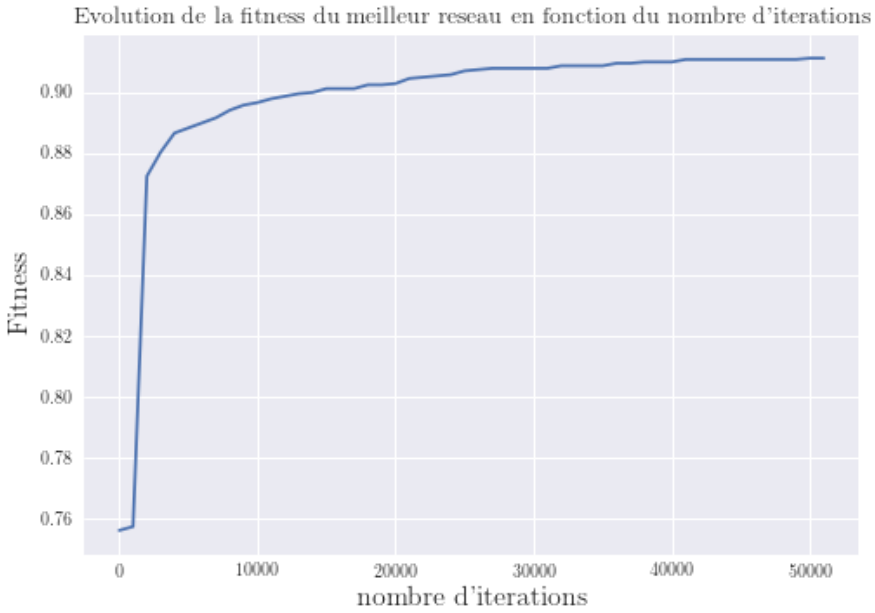


FIGURE 3.3: Evolution de la fitness en termes de générations pour la tâche L et R.

	$Q_{real} \pm$ SE	$Q_{rand} \pm$ SE	$Q_{max} \pm$ SE	$Q_m \pm$ SE
Buts fixes	0.40 ± 0.03	0.41 ± 0.05	0.69 ± 0.01	-0.04 ± 0.09
Buts variants	0.40 ± 0.03	0.41 ± 0.05	0.69 ± 0.01	-0.04 ± 0.09

TABLE 3.5: Mesure de modularité des réseaux évolués sous un but fixe ou variant.

	$Q_{real} \pm$ SE	$Q_{rand} \pm$ SE	$Q_{max} \pm$ SE	$Q_m \pm$ SE
Buts fixes	0.40 ± 0.01	0.36 ± 0.00	0.62 ± 0.01	0.15 ± 0.02
Buts variants	0.45 ± 0.00	0.36 ± 0.00	0.62 ± 0.01	0.35 ± 0.02

TABLE 3.6: Mesure de modularité des réseaux évolués sous un but fixe ou variant de l'article de référence [1].

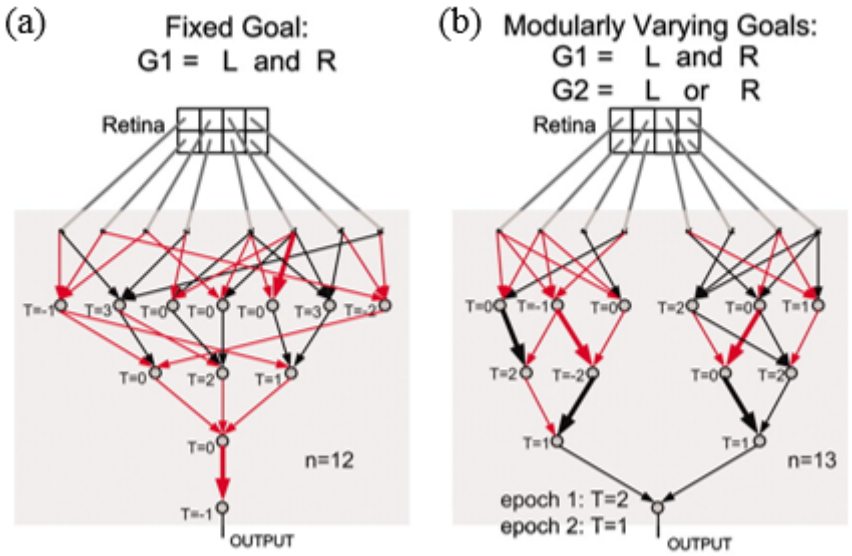


FIGURE 3.4: Représentation d'un réseau évolué sous un but fixe (a) et sous des buts variants (b). Les flèches noirs (ou rouge) représentent des connexions avec des poids positifs (resp. négatifs). Les flèches plus épaisses signifie une double connexion. n est le nombre de neurones dans le réseau et T est le seuil de ces neurones. Source :[1]

3.5 Conclusion

Dans ce chapitre, nous avons donné une description du problème qui nous concernait. Nous avons également appliqué différentes notions que nous avons développées dans les chapitres précédents, à savoir les réseaux de neurones artificiels, la mesure de modularité et les algorithmes génétiques. De plus, nous avons aussi analysé les résultats que nous avons obtenus. Nous avons conclu que contrairement aux résultats de l'article, faire évoluer les réseaux de neurones sous un but fixe ou changeant n'avait pas impacter la structure des réseaux, les uns ne sont pas plus modulaire que les autres.

Discussion

En biologie, il ressort de la plupart des avis que les organismes possèdent une organisation modulaire. Cependant, en évolution artificielle, il est difficile de maintenir et d'évoluer vers une telle structure. Cela est dû en partie au fait qu'il est difficile d'expliquer son origine. La modularité est une notion abstraite qui n'interagit pas directement avec son environnement. Il n'est donc pas évident de donner une explication de l'évolution de la modularité.

Nous avons vu que N. Kashtan et U. Alon donnent comme explication possible que la modularité apparait lorsque l'évolution se fait sous un environnement changeant au cours du temps. Cette idée vient du fait que plus un organisme est modulaire, plus il serait capable de s'adapter et de survivre dans un environnement changeant. Malheureusement nous n'avons pas obtenu les mêmes résultats que dans l'article.

D'autres procédés ont également été mis en place afin de faire apparaître la modularité. L'une d'elles (tirée de l'article de Wagner, Pavlicev et Cheverud [17]) est la méthode de duplication-différentiation. Cette méthode consiste à dupliquer un noeud du réseau. Ce nouveau noeud hérite également des connexions de son noeud parent. Il est ensuite possible que le noeud perde une connexion ou au contraire en obtienne une nouvelle. Les seuls paramètres que contiennent cette méthode sont donc le taux de suppression et d'ajout de connexion. La modularité n'apparait que pour une petite gamme possible de ces paramètres. Dans ce procédé, l'origine de la modularité est favorisée par la mutation mais le processus de sélection est également nécessaire.

Aujourd'hui encore, l'évolution de la modularité est un processus qui n'est pas encore bien compris. De nombreuses études sont réalisées afin d'essayer d'expliquer l'évolution de la modularité.

Bibliographie

[1] Kashtan, N. et Alon, U., *Spontaneous evolution of modularity and network motifs*, PNAS, vol. 102, no. 39, 13773-13778, 2005.

[2] Laloux P., *Neurophysiologie et système nerveux central - Les mémoires*, Université de Namur, 2018.

[3] Dr Alice Roberts, *Le grand guide visuel du corps humain*, ERPI édition PEARSON, p.64,298-301, 307, 2011.

[4] Raven et al, *Biologie* 2^e *édition*, de boeck supérieur, p.889, 2011.

[5] Rojas R., *Neural Networks*, Springer-Verlag, Berlin, 1996.

[6] Nicolay D., *Modélisation et apprentissage des réseaux de neurones artificiels* , mémoire, Université de Namur, 2011.

[7] Newman M. E. J. et Girvan M., *Finding and evaluating community structure in networks* , Physical Review E 69(2 Pt 2) :026113, 2004.

[8] Newman M. E. J. et Girvan M., *Community structure in social and biological networks* , PNAS, vol. 99, no.12, 7821–7826, 2002.

[9] Newman M. E. J.,*Scientific collaboration networks. II. Shortest paths, weighted networks, and centrality*, PHYSICAL REVIEW E, VOLUME 64, 016132, 2001.

[10] Newman M. E. J.,*Fast algorithm for detecting community structure in networks*, PHYSICAL REVIEW E, VOLUME 69, 066133, 2004.

[11] Freeman L. C. ,*A set of measures of centrality based on betweenness*, Sociometry, 40, 35-41, 1977.

[12] Blondel V. D. , Guillaume J-L, Lambiotte R., Lefebvre E., *Fast unfolding of communities in large networks*, J. Stat. Mech., P10008, 1742-5468, 2008.

[13] Piazza S., *Algorithmes génétiques et leurs applications aux réseaux de neurones*, mémoire, Université de Namur, 2003.

[14] Collet P., *MoocOSE sur l'optimisation Stochastique évolutionnaire*, Université de Strasbourg, 2014.

[15] Hendrickx R., *Les agorithmes génétiques : théorie et applications*, mémoire, Université de Namur, 2011.

[16] *Le Petit Larousse illustré*, Larousse, Paris, 2007.

[17] Wagner G. P., Pavlicev M. et Cheverud J. M., *The road to modularity*, Nature Reviews Genetics, 8(12) :921–931, 2007.

Annexe A

Méthode de Newman et Girvan

```
% Newman–Girvan community finding algorithm
% source: Newman, M.E.J., Girvan, M., "Finding and
% evaluating community structure in networks"
% Algorithm idea:
% 1. Calculate betweenness scores for all edges in the
%    network.
% 2. Find the edge with the highest score and remove it
%    from the network.
% 3. Recalculate betweenness for all remaining edges.
% 4. Repeat from step 2.
% INPUTs: adjacency matrix (adj), number of modules (k)
% OUTPUTs: modules (components) and modules history -
%          each "current" module, Q - modularity metric
% Other routines used: edge_betweenness.m, subgraph.m,
%                      numedges.m
% GB, April 17, 2006, computing modularity,
% added April 28, 2011
```

```
function [modules,module_hist,Q] = newmangirvan(adj,k)

n=size(adj,1);
module_hist{1} = [1:n]; % current component
modules{1}=[1:n];

curr_mod=1;

adj_temp=adj;

while length(modules)<k

    w=edge_betweenness(adj_temp);
    % need to remove el(indmax,:)
    [wmax,indmax]=max(w(:,3));
    adj_temp(w(indmax,1),w(indmax,2))=0;
    % symmetrize
    adj_temp(w(indmax,2),w(indmax,1))=0;

    % keep on removing edges
    if isconnected(adj_temp); continue; end

    comp_mat = find_conn_comp(adj_temp);
    for c=1:length(comp_mat)
        modules{length(modules)+1}=modules{curr_mod}(
        comp_mat{c});
    end

    % remove "now" disconnected component (curr_mod)
    from modules
    modules{curr_mod}=modules{1};
    modules=modules(2:length(modules));

    modL=[];
    for j=1:length(modules)
        modL(j)=length(modules{j})
    end
    [maxL,indL]=max(modL);
    curr_mod=indL;

    module_hist{length(module_hist)+1}=modules{indL};
    adj_temp=subgraph(adj,modules{indL});

end % end of while loop
```

```
% Computing the modularity for the final module break-
% down.
% Defined as: Q=sum_over_modules_i (eii-ai^2) (eq 5) in
% Newman and Girvan.
% eij = fraction of edges that connect community i to
% community j
% ai=sum_j (eij)

% compute the total number of edges
nedges=numedges(adj);

Q = 0;
for m=1:length(modules)
   module=modules{m};
   adj_m=subgraph(adj,module);

   e_mm=numedges(adj_m)/nedges;
   a_m=sum(sum(adj(module,:)))/(2*nedges);

   Q = Q + (e_mm - a_m^2);
end
end
```

```
% Edge betweenness routine, based on shortest paths
% INPUTs: edgelist, mx3, m - number of edges
% OUTPUTs: w - betweenness per edge
% Note: Valid for undirected graphs only
% Source: Newman, Girvan, "Finding and evaluating
%         community structure in networks"
% Other routines used: adj2edgeL.m, numnodes.m,
%                      numedges.m, kneighbors.m
% Last modified: November 14, 2009

function ew = edge_betweenness(adj)

el=adj2edgeL(adj);  % the corresponding edgelist
n = numnodes(adj);  % number of nodes
m = numedges(adj);  % number of edges

ew = zeros(size(el,1),3); % edge betweenness - output

for s=1:n % across all (source) nodes

    % compute the distances and weights starting at
    % source node i
    d=inf(n,1); w=inf(n,1);
    d(s)=0; w(s)=1; % source node distance and weight
    queue=[s];      % add to queue
    visited=[];

    while not(isempty(queue))

        j=queue(1); % pop first member
        visited=[visited j];
        % find all adjacent nodes, 1 step away
        neigh=kneighbors(adj,j,1);

        for x=1:length(neigh)
            % add to queue if unvisited
            nei=neigh(x);

            if isempty(find(visited==nei)) & isempty(
            find(queue==nei))
                queue=[queue nei]
            end

        end
```

```
    for x=1:length(neigh)
        nei=neigh(x);
        if d(nei)==inf    % not assigned yet
            d(nei)=1+d(j);
            w(nei)=w(j);
        elseif d(nei)<inf & d(nei)==d(j)+1
        % assigned already, add the new path
            w(nei)=w(nei)+w(j);
        elseif d(nei)<inf & d(nei)<d(j)+1
            'do_nothing';
        end
    end
    % remove the first element
    queue=queue(2:length(queue));
end

% edge betweenness for every source node (iteration)
eww = zeros(size(el,1),3);

% find every leaf - no path from "s" to other
% vertices goes through the leaf
% farthest away from source
leaves = find(d==max(d));
for l=1:length(leaves)
    leaf=leaves(l);
    neigh=kneighbors(adj,leaf,1);
    nei2rem=[];
    for x=1:length(neigh)
        if isempty(find(leaves==neigh(x)))
            nei2rem=[nei2rem neigh(x)]
        end
    end
    % remove other leaves among the neighbors
    neigh=nei2rem;
    for x=1:length(neigh)
        indi=find(el(:,1)==neigh(x));
        indj=find(el(:,2)==leaf);
        % should be only one element at the
        % intersection
        indij=intersect(indi,indj);
        eww(indij,3)=w(neigh(x))/w(leaf);
    end
end
```

```
    dsort=unique(d);
    % reverse sort of unique distance values
    dsort=-sort(-dsort);

    for x=1:length(dsort)
        leaves=find(d==dsort(x));
        for l=1:length(leaves)
            leaf=leaves(l);
            neigh=kneighbors(adj,leaf,1);
            up_neigh=[]; down_neigh=[];
            for x=1:length(neigh)
                if d(neigh(x))<d(leaf)
                    up_neigh=[up_neigh neigh(x)];
                elseif d(neigh(x))>d(leaf)
                    down_neigh=[down_neigh neigh(x)];
                end
            end
            sum_down_edges=0;
            for x=1:length(down_neigh)
                indi=find(el(:,1)==leaf);
                indj=find(el(:,2)==down_neigh(x));
                indij=intersect(indi,indj);
                sum_down_edges=sum_down_edges+eww(indij
                ,3);
            end
            for x=1:length(up_neigh)
                indi=find(el(:,1)==up_neigh(x));
                indj=find(el(:,2)==leaf);
                indij=intersect(indi,indj);
                eww(indij,3)=w(up_neigh(x))/w(leaf)*(1+
                sum_down_edges);
            end
        end
    end
    for e=1:size(ew,1); ew(e,3)=ew(e,3)+eww(e,3); end
end

for e=1:size(ew,1)
    ew(e,1)=el(e,1);
    ew(e,2)=el(e,2);
    ew(e,3)=ew(e,3)/n/(n-1);    % normalize by the total
    number of paths
end
```

Annexe B

Méthode de Louvain

```
function [M,Q]=community_louvain(W,gamma,M0,B)

% COMMUNITY_LOUVAIN       Optimal community structure
%
% M      = community_louvain(W);
% [M,Q] = community_louvain(W,gamma);
% [M,Q] = community_louvain(W,gamma,M0);
% [M,Q] = community_louvain(W,gamma,M0,'potts');
% [M,Q] = community_louvain(W,gamma,M0,'negative_asym');
% [M,Q] = community_louvain(W,[],[],B);
%
% The optimal community structure is a subdivision of
% the network into nonoverlapping groups of nodes which
% maximizes the number of within-group edges, and
% minimizes the number of between-group edges.
%
% This function is a fast and accurate multi-iterative
% generalization of the Louvain community detection
% algorithm. This function subsumes and improves upon,
%  modularity_louvain_und.m, modularity_finetune_und.m,
%  modularity_louvain_dir.m, modularity_finetune_dir.m,
%  modularity_louvain_und_sign.m
% and additionally allows to optimize other objective
% functions (includes built-in Potts-model Hamiltonian,
% allows for custom objective-function matrices).
%
% Inputs:
%    W,
%          directed/undirected weighted/binary connection
%          matrix with positive and possibly negative
%          weights.
```

```
%     gamma,
%           resolution parameter (optional)
%                gamma>1,          detects smaller modules
%                0<=gamma<1,       detects larger modules
%                gamma=1,          classic modularity (default)
%     M0,
%           initial community affiliation vector (optional)
%     B,
%           objective-function type or custom objective
%           matrix (optional)
%            'modularity',        modularity (default)
%            'potts',             Potts-model Hamiltonian (for
%                                 binary networks)
%            'negative_sym',      symmetric treatment of
%                                 negative weights
%            'negative_asym',     asymmetric treatment of
%                                 negative weights
%             B,                  custom objective-function
%                                 matrix
%
%           Note: see Rubinov and Sporns (2011) for a
%           discussion of symmetric vs. asymmetric
%           treatment of negative weights.
%
% Outputs:
%     M,
%           community affiliation vector
%     Q,
%           optimized community-structure statistic
%           (modularity by default)
%
% Example:
%     % Iterative community finetuning.
%     % W is the input connection matrix.
%     n  = size(W,1);           % number of nodes
%     M  = 1:n;                 % initial community
%                                 affiliations
%     Q0 = -1; Q1 = 0;          % initialize modularity
%                                 values
%     while Q1-Q0>1e-5;         % while modularity
%                                 increases
%           Q0 = Q1;            % perform community
%                                 detection
%           [M, Q1] = community_louvain(W, [], M);
```

```
%     end
%
% References:
%     Blondel et al. (2008)  J. Stat. Mech. P10008.
%     Reichardt and Bornholdt (2006) Phys. Rev. E 74,
%     016110.
%     Ronhovde and Nussinov (2008) Phys. Rev. E 80,
%     016109
%     Sun et al. (2008) Europhysics Lett 86, 28004.
%     Rubinov and Sporns (2011) Neuroimage 56:2068-79.
%
% Mika Rubinov, U Cambridge 2015-2016

% Modification history
% 2015: Original
% 2016: Included generalization for negative weights.
%       Enforced binary network input for Potts-model
%       Hamiltonian.
%       Streamlined code and expanded documentation.

W=double(W);                    % convert to double format
n=length(W);                    % get number of nodes
s=sum(sum(W));                  % get sum of edges

if ~exist('B','var') || isempty(B)
    type_B = 'modularity';
elseif ischar(B)
    type_B = B;
else
    type_B = 0;
    if exist('gamma','var') && ~isempty(gamma)
        warning('Value of gamma is ignored in
        generalized mode.')
    end
end
if ~exist('gamma','var') || isempty(gamma)
    gamma = 1;
end

if strcmp(type_B,'negative_sym') || strcmp(type_B,'
negative_asym')
    W0 = W.*(W>0);              % positive weights matrix
    s0 = sum(sum(W0));          % weight of positive links
    B0 = W0-gamma*(sum(W0,2)*sum(W0,1))/s0;
```

```
                                % positive modularity

    W1 =-W.*(W<0);              % negative weights matrix
    s1 = sum(sum(W1));          % weight of negative links
    if s1                       % negative modularity
        B1 = W1-gamma*(sum(W1,2)*sum(W1,1))/s1;
    else
        B1 = 0;
    end
elseif min(min(W))<-1e-10
    err_string = [
        'The input connection matrix contains negative
        weights.\nSpecify ' ...
        '''negative_sym'' or ''negative_asym'' objective
        -function types.'];
    error(sprintf(err_string))                  %ok<SPERR>
end
if strcmp(type_B,'potts') && any(any(W ~= logical(W)))
    error('Potts-model Hamiltonian requires a binary W.'
    )
end

if type_B
    switch type_B
        case 'modularity';
            B = (W-gamma*(sum(W,2)*sum(W,1))/s)/s;
        case 'potts';
            B =  W-gamma*(~W);
        case 'negative_sym';
            B = B0/(s0+s1) - B1/(s0+s1);
        case 'negative_asym';
            B = B0/s0      - B1/(s0+s1);
        otherwise;
            error('Unknown objective function.');
    end
else    % custom objective function matrix as input
    B = double(B);
    if ~isequal(size(W),size(B))
        error('W and B must have the same size.')
    end
end
if ~exist('M0','var') || isempty(M0)
    M0=1:n;
elseif numel(M0)~=n
```

```
    error('M0 must contain n elements.')
end

[~,~,Mb] = unique(M0);
M = Mb;

B = (B+B.')/2;              % symmetrize modularity matrix
Hnm=zeros(n,n);             % node-to-module degree
for m=1:max(Mb)             % loop over modules
    Hnm(:,m)=sum(B(:,Mb==m),2);
end

Q0 = -inf;
% compute modularity
Q = sum(B(bsxfun(@eq,M0,M0.')));
first_iteration = true;
while Q-Q0>1e-10
    % flag for within-hierarchy search
    flag = true;
    while flag;
        flag = false;
        % loop over all nodes in random order
        for u=randperm(n)
            % current module of u
            ma = Mb(u);
            dQ = Hnm(u,:) - Hnm(u,ma) + B(u,u);
            % (line above) algorithm condition
            dQ(ma) = 0;

            % maximal increase in modularity and
            corresponding module
            [max_dQ,mb] = max(dQ);
            % if maximal increase is positive
            if max_dQ>1e-10;
                flag = true;
                % reassign module
                Mb(u) = mb;

                % change node-to-module strengths
                Hnm(:,mb) = Hnm(:,mb)+B(:,u);
                Hnm(:,ma) = Hnm(:,ma)-B(:,u);
            end
        end
    end
```

```
    % new module assignments
    [~,~,Mb] = unique(Mb);

    M0 = M;
    if first_iteration
        M=Mb;
        first_iteration=false;
    else
        % loop through initial module assignments
        for u=1:n
            % assign new modules
            M(M0==u)=Mb(u);
        end
    end

    n=max(Mb);          % new number of modules
    B1=zeros(n);        % new weighted matrix
    for u=1:n
        for v=u:n
            % pool weights of nodes in same module
            bm=sum(sum(B(Mb==u,Mb==v)));
            B1(u,v)=bm;
            B1(v,u)=bm;
        end
    end
    B=B1;

    Mb=1:n;             % initial module assignments
    Hnm=B;              % node-to-module strength

    Q0=Q;
    Q=trace(B);         % compute modularity
end
```