LES RESEAUX DE NEURONES ARTIFICIELS

Introduction au connexionnisme

- Seconde édition -

Claude Touzet

Du même auteur

disponible sur : www.machotte.fr

C. Touzet, *Hypnose, sommeil, placebo? Les réponses de la Théorie neuronale de la Cognition – tome 2*, 168 pages, éd. la Machotte, 2014, ISBN 978-2-919411-02-3.

C. Touzet, *Conscience, intelligence, libre-arbitre ? Les réponses de la Théorie neuronale de la Cognition*, 156 pages, éd. la Machotte, 2010, ISBN 978-2-919411-00-9.

Les réseaux de neurones artificiels ~

Introduction au connexionnisme - Seconde édition

Claude Touzet

Editions la Machotte
155 impasse d'Où Pèbre, F–13390 AURIOL, France (*www.machotte.fr*)

Remerciements

Il y a 25 ans, je terminais la rédaction du second ouvrage en langue française sur les réseaux de neurones artificiels[1]. L'éditeur fit faillite peu après sa parution, ce qui me permis d'en diffuser gratuitement la version électronique (pdf). De ce fait, ce livre connu un certain succès auprès des étudiants.

Aujourd'hui, avec le recul que confèrent 30 années d'expérience dans la compréhension du cerveau et la mise en œuvre de réseaux de neurones artificiels, je pense qu'il est temps de proposer une version complètement mise à jour de cette « introduction au connexionnisme ». Je me dois d'ailleurs de remercier mes nombreux étudiants et élèves-ingénieurs qui – tant par leurs questions que par leurs copies d'examen – m'ont aidé à cerner les points délicats de cette discipline.

1 C. Touzet, *Les réseaux de neurones artificiels : introduction au connexionnisme*, 150 pages, Préface de Jeanny Hérault, EC2 éd., Paris, 1992.

Table des matières

Avertissement

Avant de plonger dans le vif du sujet, je voudrai assurer le lecteur de ma conviction qu'à court terme (10 ans) une nouvelle industrie va émerger – de la taille de l'industrie automobile.

La Théorie neuronale de la Cognition[2] (TnC) fait de la colonne corticale l'unité de traitement de l'information par le cerveau. Une colonne corticale agrège environ 100 000 neurones. On compte seulement 160 000 colonnes corticales dans le cortex, réparties au sein d'environ 160 cartes corticales – dont la moitié déjà sont connues.

La TnC explique suffisamment précisément le fonctionnement du cerveau pour que l'on puisse le simuler complètement – et donc construire un robot intelligent et conscient. Nous aurons donc bientôt, en plus d'une voiture au garage, un robot à la maison...

Dix pour cent de notre économie sera liée à cette nouvelle robotique, et les ingénieurs qui construiront les cerveaux artificiels devront comprendre et manipuler les réseaux de neurones artificiels. Il paraît que la moitié des jobs de 2030 ne sont pas encore inventés – en voici quelques uns !

Bonne lecture

2 que j'ai présentée dans deux ouvrages à destination du grand public parus aux éditions la Machotte en 2010 et 2014.

S'agissant d'un ouvrage de vulgarisation à l'usage des étudiants de tous les âges, nous avons tenu, en nous basant sur notre expérience d'enseignant, à proposer les outils pédagogiques que sont les exercices et les travaux pratiques. Il s'agit bien entendu d'aider le lecteur à vérifier sa compréhension des concepts, des modèles et de le familiariser à la manipulation des algorithmes. Nous espérons que vous vous impliquerez dans ce "surplus" de travail proposé. Toutes les réponses se trouvent évidemment dans ce livre.

« Les réseaux de neurones artificiels (RNA) sont
des réseaux fortement connectés de processeurs élémentaires
fonctionnant en parallèle.
Chaque processeur élémentaire calcule une sortie unique
sur la base des informations qu'il reçoit.
Toute structure hiérarchique de réseaux est évidemment un réseau. »
Définition

Introduction

L'informatique est la science du traitement automatique de l'information. Son développement est souvent confondu avec celui des machines de traitement : les ordinateurs. Depuis les débuts (ENIAC 1946) jusqu'à aujourd'hui, les ordinateurs sont de plus en plus puissants.

Cependant, cette augmentation de puissance ne permet pas de toujours résoudre les problèmes d'une application informatique dans un domaine particulier. L'idée s'est donc installée que ce n'était peut être pas tant le matériel que le logiciel qui pêchait par manque de puissance. Le développement logiciel s'appuie sur plusieurs approches. Deux parmi les plus utilisées sont l'approche algorithmique et l'approche basée sur la connaissance.

1. Une approche algorithmique nécessite l'écriture (avant la transcription dans un quelconque langage de programmation) du processus à suivre pour résoudre le problème. Lorsque le problème est complexe, ce peut être une étape coûteuse ou impossible. D'autre part, les ordinateurs sont des machines complètement logiques (et même binaires) qui suivent à la lettre chacune des instructions du programme. C'est un avantage lorsque tous les cas ont été prévus à l'avance par l'algorithmicien. Ce n'est hélas pas toujours possible. Dans ce cas, dixit l'informaticien : "*c'est une faute de la machine*". Rien de plus faux ! Ainsi les systèmes informatiques embarqués (à bord des avions, de la navette spatiale, etc.) tentent de pallier à ce manque (prévisible) de clairvoyance de l'algorithmicien en triplant les logiciels, chacun étant développés indépendamment par une équipe différente, dans des langages différents. Les risques de laisser l'ordinateur aux prises avec une situation imprévue, où son comportement ne serait pas adapté, sont ainsi considérablement réduits. Rappelons nous le haro lancé sur les programmes boursiers lors de la chute de la bourse en 1987.

2. La seconde approche possible est celle de l'intelligence artificielle (appelée IA par commodité), avec pour applications les plus connues les systèmes experts. Ici, la résolution du problème est confiée à un ensemble de règles données par l'expert humain du domaine. Ici encore toutes les règles doivent avoir été exprimées préalablement au traitement, et le programme demeure binaire dans son exécution. Les cas qui n'ont pas été prévus par l'expert ne seront pas correctement traités. L'introduction de la logique floue ne change pas la nature des limitations d'emploi du programme : l'exécution reste totalement déterministe. En fait, l'approche basée sur la connaissance se limite à des domaines d'application où la modélisation de la connaissance, par exemple sous forme de règles, est possible. Ces domaines sont souvent ceux des sciences dites "dures" comme l'électronique, la mécanique, la physique, etc. – par opposition aux sciences dites "molles" comme la médecine, la psychologie, la philosophie, etc., où la connaissance est plus empirique. L'IA se révèle donc être principalement un moyen commode de stocker de la connaissance sous forme explicite.

Ces deux approches ne suffisent pas à répondre à tous les problèmes existants comme par exemple dans le domaine de la reconnaissance de formes (images ou signaux), celui du diagnostic, du contrôle moteur, de la traduction automatique, de la compréhension du langage... Tous sont depuis longtemps explorés à l'aide des approches algorithmiques (et à base de connaissances) et n'ont pas rencontré le succès escompté. Pourtant, des êtres vivants relativement simples sont capables de réaliser certaines de ces opérations apparemment sans difficulté. Il suffit pour s'en rendre compte de lever les yeux, suivre le vol de la mouche et essayer de la capturer. Que dire alors du déplacement au sonar de la chauve souris, etc. ?

Une troisième approche au traitement automatique de l'information s'offre à nous, où l'on cherche à s'inspirer du traitement de l'information effectué par le cerveau.

3. L'hypothèse principale, à la base de l'essor des RNA, est que les réseaux de neurones biologiques sont à la base des fonctions cognitives, et que nous pensons qu'en utilisant les mêmes composants (le neurone), dans la même configuration (en réseaux), alors nous sommes plus proches d'obtenir une « intelligence artificielle ». Ce qui nous intéresse réellement, c'est l'intelligence, la capacité à répondre correctement alors même que la question est nouvelle, qu'elle n'a jamais été posée auparavant. Les réseaux

de neurones artificiels s'inspirent donc de la biologie, non par obligation comme pour la neuromimétique, mais par intérêt.

I. Historique des RNA

1890 : W. James, célèbre psychologue américain (et frère du tout aussi célèbre écrivain Henry James) introduit le concept de mémoire associative, et propose ce qui deviendra une loi de fonctionnement pour l'apprentissage sur les réseaux de neurones connue plus tard sous le nom de loi de Hebb.

1943 : J. Mc Culloch et W. Pitts laissent leurs noms à une modélisation du neurone biologique (un neurone au comportement binaire). Ceux sont les premiers à montrer que des réseaux de neurones formels simples peuvent réaliser des fonctions logiques, arithmétiques et symboliques complexes (tout au moins au niveau théorique).

1949 : D. Hebb, physiologiste américain explique le conditionnement chez l'animal par les propriétés des neurones eux-mêmes. Ainsi, un conditionnement de type pavlovien tel que, nourrir tous les jours à la même heure un chien, entraîne chez cet animal la sécrétion de salive à cette heure précise même en l'absence de nourriture. La loi de modification des propriétés des connexions entre neurones qu'il propose explique en partie ce type de résultats expérimentaux. Sa « loi » a depuis été validée par les neurosciences : LTP et LTD (Long Term Potentiation, Long Term Depression)

1957 : F. Rosenblatt développe le modèle du Perceptron. Il construit le premier neuro-ordinateur basé sur ce modèle et l'applique au domaine de la reconnaissance de formes. Notons qu'à cet époque les moyens à sa disposition sont limités et c'est une prouesse technologique que de réussir à faire fonctionner correctement cette machine pendant plus de quelques minutes.

1960 : B. Widrow, un automaticien, développe le modèle Adaline (Adaptative Linear Element). Dans sa structure, le modèle ressemble au

Perceptron, cependant la loi d'apprentissage est différente. Celle-ci est à l'origine de l'algorithme de rétro-propagation de gradient très utilisé aujourd'hui avec les Perceptrons multicouches. Les réseaux de type Adaline restent utilisés de nos jours pour certaines applications particulières.

1969 : M. Minsky et S. Papert publient un ouvrage qui met en exergue les limitations théoriques du Perceptron. Limitations alors connues, notamment concernant l'impossibilité de traiter par ce modèle des problèmes non linéaires. Ils étendent implicitement ces limitations à tous modèles de réseaux de neurones artificiels. Leur objectif est atteint, il y a abandon financier des recherches dans le domaine (surtout aux U.S.A.), les chercheurs se tournent principalement vers l'IA et les systèmes à bases de règles.

1967-1982 : Toutes les recherches ne sont, bien sûr, pas interrompues. Elles se poursuivent, mais déguisées, sous le couvert de divers domaines comme : le traitement adaptatif du signal, la reconnaissance de formes, la modélisation en neurobiologie, etc. De grands noms travaillent durant cette période tels : J. Hérault, B. Ans, etc.

1977 : T. Kohonen (professeur en ingénierie finlandais) propose une modélisation de la carte corticale. Son modèle baptisé « carte auto-organisatrice » ou « carte de Kohonen » est à la base de milliers d'applications. La carte auto-organisatrice est aussi la brique élémentaire pour la Théorie neuronale de la Cognition (2010). Enfin, la carte de Kohonen apparaîtra aussi au sein des réseaux de neurones profonds (2013).

1982 : J. J. Hopfield est un physicien reconnu à qui l'on doit le renouveau d'intérêt pour les réseaux de neurones artificiels. A cela plusieurs raisons : Au travers d'un article court, clair et bien écrit, il présente une théorie du fonctionnement et des possibilités des réseaux de neurones. Il faut remarquer la présentation anticonformiste de son article. Alors que les auteurs s'acharnent jusqu'alors à proposer une structure et une loi d'apprentissage, puis à étudier les propriétés émergentes ; J. J. Hopfield fixe préalablement le comportement à atteindre pour son modèle et construit à partir de là, la structure et la loi d'apprentissage correspondant au résultat escompté. Ce modèle est aujourd'hui encore très utilisé pour des problèmes d'optimisation. D'autre part, entre les mains de ce physicien distingué, la théorie des réseaux de neurones devient respec-

table. Elle n'est plus l'apanage d'un certain nombre de psychologues et neurobiologistes hors du coup. Enfin, une petite phrase, placée en commentaire dans son article initial, met en avant l'isomorphisme de son modèle avec le modèle d'Ising (modèle des verres de spins). Cette idée va drainer un flot de physiciens vers les réseaux de neurones artificiels. Notons qu'à cette date, l'IA est l'objet d'une certaine désillusion, elle n'a pas répondue à toutes les attentes et s'est même heurtée à de sérieuses limitations. Aussi, bien que les limitations du Perceptron mise en avant par M. Minsky ne soient pas levées par le modèle d'Hopfield, les recherches sont relancées.

1983 : La Machine de Boltzmann est le premier modèle connu apte à traiter de manière satisfaisante les limitations recensées dans le cas du Perceptron. Mais l'utilisation pratique s'avère difficile, la convergence de l'algorithme étant extrêmement longue (les temps de calcul sont considérables).

1985 : La rétro-propagation de gradient apparaît. C'est un algorithme d'apprentissage adapté aux réseaux de neurones multicouches (aussi appelés Perceptrons multicouches). Sa découverte réalisée par trois groupes de chercheurs indépendants indique que "la chose était dans l'air". Dès cette découverte, nous avons la possibilité de réaliser une fonction non linéaire d'entrée/sortie sur un réseau en décomposant cette fonction en une suite d'étapes linéairement séparables.

1990 : L'apprentissage par renforcement devient incontournable dans le domaine de l'apprentissage en robotique. Le coût du développement d'une application « RNA » est fortement réduit puisqu'il n'y a plus besoin de construire de bases d'apprentissages. Une simple fonction de renforcement – capable de qualifier (et non quantifier) la réponse du RNA – suffit.

1996 : Diverses implantations (sur cartes auto-organisatrices notamment – Q-Kohon) du renforcement voient le jour, et des propositions sont faites pour faciliter l'écriture de la fonction de renforcement.

2006 : Pour la première fois deux cartes auto-organisatrices travaillent en synergie, et permettent la synthèse instantanée de comportements en robotique.

2010 : La Théorie neuronale de la Cognition (TnC) explique comment une hiérarchie de cartes auto-organisatrices est capable de réaliser toutes

les fonctions cognitives : attention, mémoire, planification, intelligence, conscience, etc.

2013 : Les réseaux de neurones « profonds » sont les successeurs des réseaux multicouches (ils vont jusqu'à 10 couches quand leur prédécesseurs devaient s'arrêter à 2 ou 3 maxi). Les réseaux profonds obtiennent d'excellentes performances pour la classification d'images et de sons. Pour les images, ces performances sont supérieures à celles d'un humain. Pour le son, on estime aujourd'hui que la reconnaissance de la parole est un problème résolu – c'est pourquoi l'on trouve (enfin) des applications capable de comprendre ce que nous disons (exemple Siri d'Apple).

2014 : Le palimpseste learning est proposé. C'est l'aboutissement de l'évolution entamée par l'apprentissage par renforcement : il n'y a plus de fonction de renforcement ! Cet apprentissage est basé sur la loi de Hebb, mais ajoute la prise en compte du temps nécessaire à la modification d'efficacité synaptique. Le palimpseste learning trouve « tout seul » les états d 'équilibre (homéostasie). Il est particulièrement bien adapté à la commande automatique (boucle de régulation).

2015 : Les géants du logiciel (Google, FaceBook, IBM, etc.) ont recrutés pratiquement tous les experts (universitaires) en réseaux de neurones profonds pour développer de nouvelles applications. Nous pouvons donc estimer que les RNA ont répondu à nos attentes et sont en passe de résoudre certains des challenges de l'IA.

La chronologie n'est pas le seul fil conducteur que nous pouvons suivre pour comprendre et classer les divers modèles de RNA. Pour ma part, je trouve le « temps ingénieur nécessaire pour développer une application » plus instructif. Cette approche permet de séparer trois groupes selon la complexité associée à la construction de la base d'apprentissage (depuis le plus dispendieux en temps ingénieur) :

- apprentissage supervisé (Perceptron, réseaux multicouches, etc.),
- apprentissage par renforcement (Q-Kohon, TD-λ),
- auto-organisation (carte de Kohonen).

L'histoire montre que l'approche supervisée a réussi certains challenges. C'est à mon avis la moins élégante – mais la plus aisée à comprendre pour l'ingénieur. Les applications que ne peuvent pas résoudre l'approche supervisée seront *de facto* du domaine du renforce-

ment ou de l'auto-organisation. Bien que moins gourmande en « temps ingénieur », ces approches nécessitent un investissement en terme de compétence par l'ingénieur.

Ce qu'il faut retenir :

L'histoire des RNA débute il y a 120 ans. Certaines des applications envisagées par les informaticiens des années 1960 sont devenues réalités 50 ans plus tard. Pour les autres « rêves » comme l'intelligence ou la conscience artificielle, un peu de travail est encore nécessaire.

1. Le neurone biologique

Le neurone est une cellule. Le corps humain compte dix mille milliards de cellules, dont 82 milliards de cellules nerveuses ou neurones (on dit « un » neurone). Le neurone est une cellule très particulière qui consomme environ 10 fois plus d'énergie qu'une cellule normale.

D'autre part, le neurone n'est pas remplacé : c'est la même cellule de la naissance à notre décès pour cause d'âge très avancé (une cellule de la peau est remplacée chaque 3-4 semaines, un globule blanc a une durée de vie de quelques jours, etc.).

Enfin, le neurone est une abstraction car il existe plusieurs dizaines de types de neurones (neurones pyramidaux, en corbeille, en étoile, etc.) dont les tailles et les formes sont très différentes. Cependant, tous les neurones partagent le même comportement : ils transmettent des potentiels d'action à distance dont l'effet sur le neurone suivant sera excitateur ou inhibiteur vis à vis de la création d'un potentiel d'action (pour ce neurone). L'efficacité de l'effet excitateur, ou inhibiteur, est modulée par l'usage (loi de Hebb) et les conditions locales (homéostasie).

Le neurone reçoit une information via ses dendrites, et si l'activation locale est à cet instant suffisante, alors il l'a transmet via son axone à d'autres neurones. Les jonctions entre les neurones « amont » (précédents) qui alimentent ses dendrites ou « aval » (suivants) alimentées par son axone, sont appelées des synapses. Il y en a des centaines, voir des milliers pour un seul neurone. De ce fait, la simulation précise de chaque neurone d'un cerveau de taille humaine avec toutes ses synapses se heurte à des chiffres colossaux : 100 milliards de neurones x 1000 synapses = 100 billards de synapses (10^{14}, c'est à dire pour les « informaticien » : 100 Tera).

Nos simulations actuelles de RNA ne peuvent pas s'approcher de tels chiffres. Heureusement, ceci n'est pas nécessaire pour obtenir des

comportements « intelligents » de la part de nos programmes informatiques.

Notons que ce grand nombre de neurones et de synapses est un puissant indicateur que la redondance du système nerveux est importante. Faire disparaître « un » neurone n'entraîne aucun effet visible. En fait, les malades Alzheimer montrent parfois une dégénérescence allant jusqu'à 70% de perte neuronale dans certaines régions du cerveau avant que leur maladie ne soit diagnostiquée. Ceci témoigne à la fois de la forte redondance, et aussi de la plasticité du système, capable de se reconfigurer en permanence de manière à assurer sa fonction quelles que soient les lésions.

ATTENTION !

L'idée d'une fonction assurée par certaines régions neuronales, certains réseaux de neurones, voir certains neurones, pour aussi logique et attrayante qu'elle soit, est une idée fausse ! Nous cherchons (et trouvons) des fonctions parce que nous sommes programmés à rechercher une solution à un problème complexe (et le fonctionnement du cerveau en est un) par une décomposition en sous-problèmes. Ayant repéré des organisations particulières dans le cerveau (cortex, cervelet, thalamus, etc.) alors nous sommes tentés (obligés) de leur attribuer une fonction.

Pourtant, de nombreux faits viennent contredire nos tentatives dans ce domaine. Par exemple, certaines personnes présentent une cognition normale alors qu'elles n'ont qu'un unique hémisphère, d'autres – à la suite d'un AVC par exemple – vont réapprendre de nombreuses compétences en utilisant des structures neuronales différentes des aires habituelles.

Enfin, notre cerveau est le même que nous soyons membre d'une société capitaliste du XXIème siècle, d'une tribu encore à l'âge de pierre, ou d'une des castes de la société indienne. Les compétences que nous développerons, à commencer par la langue, seront toutes spécifiques à notre environnement et très efficaces (adaptées). Pour mémoire, certaines langues n'ont pas de temps « futur » ou « passé », pour d'autres la numérosité est limitée à « 1 », « 2 » et « beaucoup », ou ont pour seules couleurs « claire » et « sombre ».

Les réseaux de neurones biologiques n'ont pas donc pas de fonctions

prédestinées, ils s'auto-organisent et le résultat de cette organisation est adapté en terme de théorie de l'évolution (survie et transmission des gènes). Ceci est une bonne nouvelle ! Il nous suffira donc de doter nos RNA de capacité d'auto-organisation pour obtenir des comportements adaptés desdits réseaux.

Le neurone

Le corps cellulaire du neurone se ramifie pour former ce que l'on nomme les dendrites. Celles-ci sont parfois si nombreuses que l'on parle alors de chevelure dendritique ou d'arborisation dendritique. C'est par les dendrites que l'information est acheminée de l'extérieur vers le soma (le corps du neurone). L'information traitée par le neurone chemine ensuite le long de l'axone (unique) pour être transmise aux autres neurones.

La transmission entre deux neurones n'est pas directe. En fait, il existe un espace intercellulaire de quelques dizaines d'Angströms (10^{-9} m) entre l'axone du neurone afférent et les dendrites (on dit une dendrite) du neurone efférent. La jonction entre deux neurones est appelée la synapse (fig. 1).

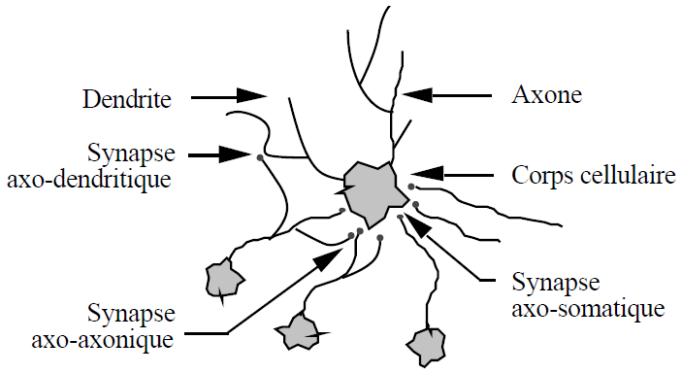


Figure 1. Un neurone avec son arborisation dendritique.

Selon le type du neurone (fig. 2), la longueur de l'axone peut varier de quelques microns à 1,50 mètres pour un moto-neurone. De même les dendrites mesurent de quelques microns à 1,50 mètres pour un neurone

sensoriel de la moelle épinière. Le nombre de synapses par neurone varie aussi considérablement de plusieurs centaines à une dizaine de milliers.

Avec 10 000 synapses reçues par neurone, une unique synapse a très peu d'effet sur le comportement du neurone. Il faut un « consensus » d'un grand nombre de synapses pour obtenir une excitation ou une inhibition de ce neurone. De fait, lorsque deux neurones sont interconnectés, ce n'est pas par une unique synapse, mais plutôt par quelques centaines ou milliers d'entre elles. Un neurone est donc connecté à quelques dizaines ou centaines d'autres neurones – au maximum !

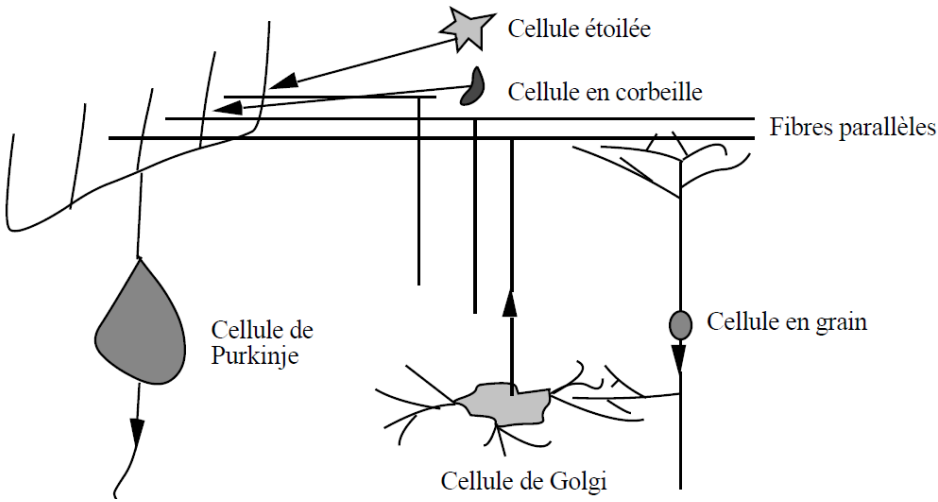


Figure 2. Description schématique des divers types structuraux de neurones présents dans le cortex cérébelleux. Les axones ont été repérés par une flèche.

Physiologie

La physiologie du neurone est liée aux propriétés de la membrane nerveuse et au métabolisme de la cellule. La différence de potentiel mesurée entre le milieu intérieur de la cellule et le milieu extérieur est de -60 mV. Pour maintenir une telle différence de potentiel, la cellule fait appel à des pompes ioniques (Na+, K+, ...). Cependant, une faible dépolarisation de la membrane entraîne une certaine perméabilité aux ions sodiums (Na+), dont l'effet peut être catastrophique au niveau cellulaire. En effet, à partir d'une certaine valeur seuil de dépolarisation de la membrane, il y a rupture des équilibres ioniques et création d'un potentiel d'action (aussi nommé "spike" en anglais, fig. 3).

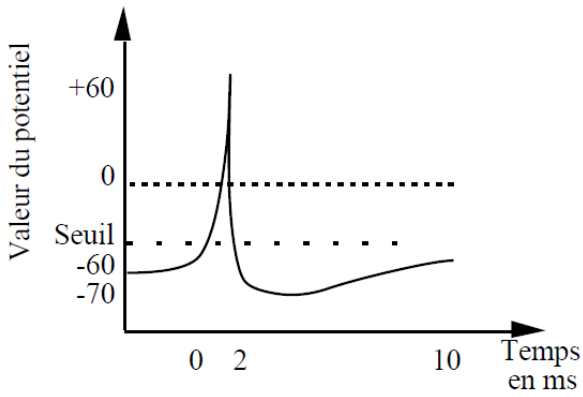


Figure 3. Un potentiel d'action.

Les ions, Na+ en particulier, s'engouffrent en nombre dans la cellule (aidés en cela par l'ouverture des canaux Na+ et une différence de potentiel très attirante de -60 mV). En une milliseconde, la différence de potentiel devient égale à +60 mV (fig. 4). En fait, à partir d'une valeur de potentiel nulle, l'équilibre ionique est établi et les ions ne devraient plus pénétrer dans la cellule. Cependant, l'effet d'entraînement est tel que cette valeur d'équilibre théorique est largement dépassée. Les différents canaux ioniques se referment alors, les pompes ioniques se remettent à fonctionner, rejetant à l'extérieur de la cellule les ions en excès. Là aussi, on constate un certain effet d'entraînement : le retour à la normale passe d'abord par une phase d'hyperpolarisation. Le potentiel de repos (-60 mV) est dépassé jusqu'à atteindre (-70 mV).

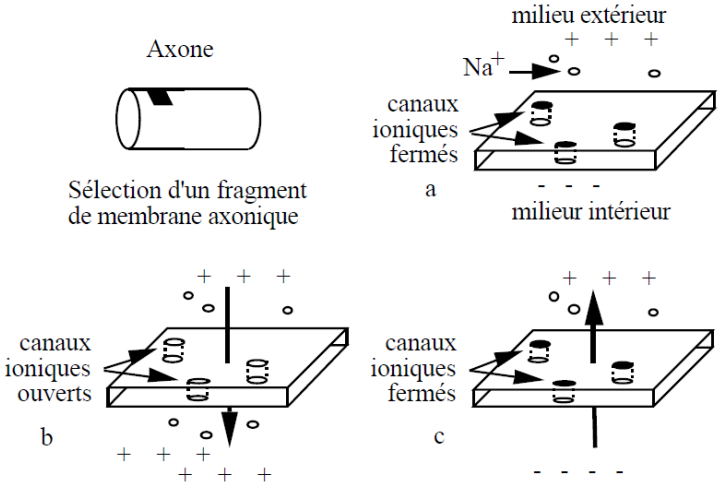


Figure 4. Passage d'un potentiel d'action au niveau de la membrane de l'axone
a) Équilibre ionique (au repos). b) Arrivée d'un potentiel d'action (dépolarisation).
c) Après la dépolarisation : l'hyperpolarisation.

Toute cette série d'événements cataclysmiques au niveau cellulaire n'aura duré que 5 à 10 millisecondes. Durant la phase d'hyperpolarisation, le neurone est très difficilement excitable. Ce qui s'explique par le fait que la différence de potentiel par rapport à la valeur seuil (S) est plus importante que celle au repos. 1.3 Création d'un potentiel d'action La dépolarisation initiale de la membrane axonique est créée par l'arrivée de potentiels d'action des neurones afférents sur les synapses dendritiques et somatiques. En fait, à l'arrivée d'un potentiel d'action sur une synapse, un neuromédiateur est libéré dans l'espace synaptique. Il va ouvrir des canaux ioniques sur la membrane post-synaptique, créant ainsi une dépolarisation (aussi appelée potentiel évoqué) qui s'étend jusqu'à l'axone (fig. 5).

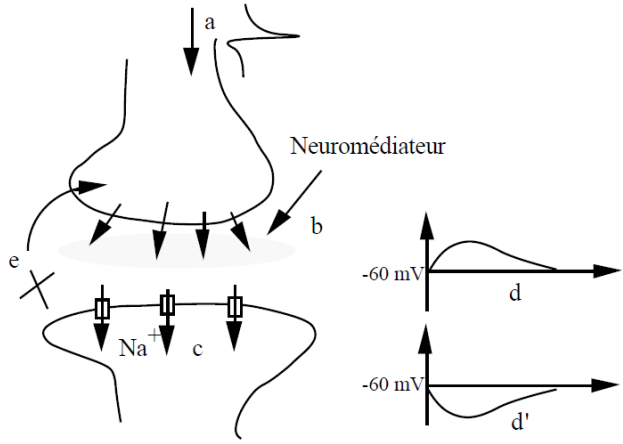


Figure 5. Fonctionnement au niveau synaptique.
a) Arrivée d'un potentiel d'action.
b) Libération du neuromédiateur dans l'espace synaptique.
c) Ouvertures des canaux ioniques dues au neuromédiateur.
d) Génération d'un potentiel évoqué excitateur.
d') Génération d'un potentiel évoqué inhibiteur. Les synapses inhibitrices empêchent la génération de potentiel d'action.
e) Fermeture des canaux, élimination ou recapture du neuromédiateur.

Les dépolarisations unitaires sont sommées dans l'espace (toutes les synapses du neurone) et dans le temps (sur une période de quelques millisecondes) et génèrent, éventuellement, un potentiel d'action sur le neurone post-synaptique. Ainsi que le montre la figure 6, la génération d'un potentiel d'action est le fruit de nombreuses dépolarisations, l'action d'une seule synapse est pratiquement sans effet.

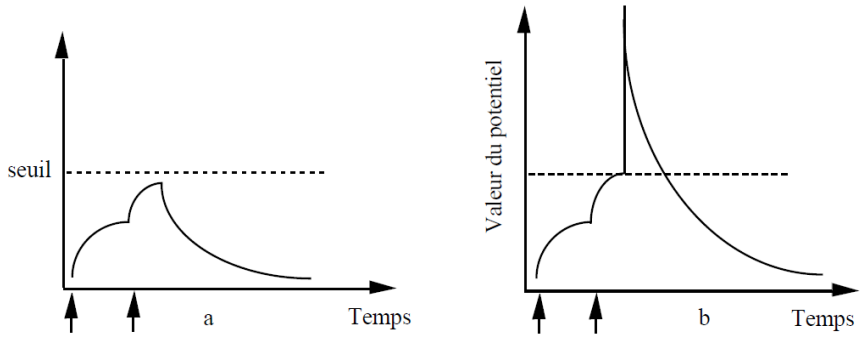


Figure 6. Sommation spatio-temporelle : addition des potentiels évoqués à la fois dans l'espace et dans le temps.
a) Les 2 potentiels évoqués (flèches) ne dépassent pas la valeur seuil.
b) Les 2 potentiels évoqués dépassent le seuil et génèrent un potentiel d'action.

Message nerveux

Le système nerveux travaille avec (entre autres) un codage en fréquence. C'est le nombre de potentiels d'action par seconde (fréquence) et les variations de fréquence (fréquence instantanée) qui codent l'information. Un potentiel d'action isolé ne signifie rien. Rappelons d'autre part que tous les potentiels d'action ont la même valeur de potentiel. Par exemple (fig. 7), les messages transmis lors de mouvements du coude permettent de connaître en fonction de la fréquence : la valeur de l'angle et en fonction des variations de fréquences : la vitesse de rotation entre deux positions.

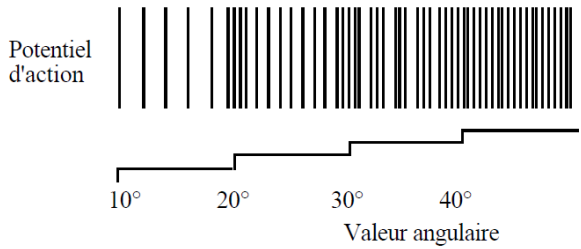


Figure 7. Codage en fréquence lors du mouvement d'une articulation.

Circuits neuronaux

Nous avons vu que chaque neurone est une unité autonome au sein du

cerveau. Le neurone reçoit en continu des entrées. Le corps cellulaire du neurone est le centre de contrôle. C'est là que les informations reçues sont interprétées. La réponse, unique, à ces signaux est envoyée au travers de l'axone. L'axone fait synapse sur d'autres neurones (un millier). Le signal transmis peut avoir un effet excitateur ou inhibiteur. Le traitement très simple réalisé par chaque neurone indique que l'information n'est pas stockée dans les neurones, mais est plutôt le résultat du comportement de toute la structure interconnectée. L'information est, principalement, dans l'architecture des connexions et dans la force de ces connexions. C'est ce que nous allons vérifier avec quelques expérimentations simples réalisées sur l'aplysie (limace de mer, fig. 8). Des modifications comportementales importantes résultent de modifications simples au niveau synaptique. Les connexions renforcent ou diminuent leur efficacité (modification des forces de connexions). Dans les cas extrêmes, de nouvelles connexions apparaissent ou disparaissent (modification de l'architecture).

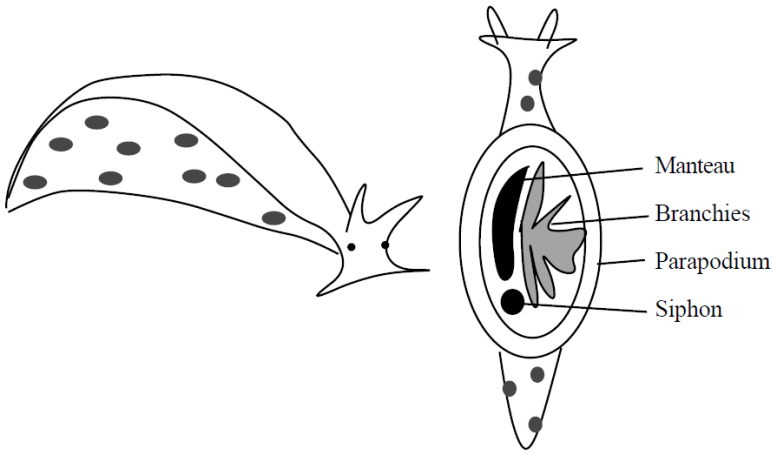


Figure 8. Aplysie ou limace de mer (abondante en Méditerranée).
Au toucher du siphon ou du manteau, la contraction du siphon entraîne le retrait des branchies sous le manteau dans un réflexe de défense.

Habituation

Description de l'expérience : Le neurone sensoriel est activé par le toucher du manteau. Le neurone moteur agit alors en rétractant les branchies (fig. 9). Lorsque la stimulation est répétée, la réponse de

l'animal devient plus faible, jusqu'à une absence de réaction au toucher. C'est le phénomène de l'habituation (fig. 10).

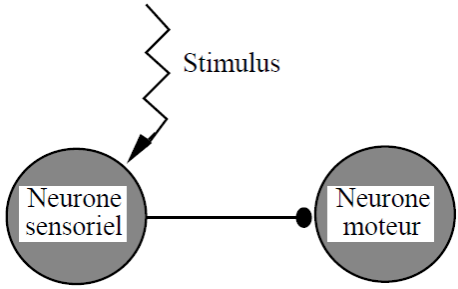


Figure 9. Circuit mis en jeu dans l'habituation.

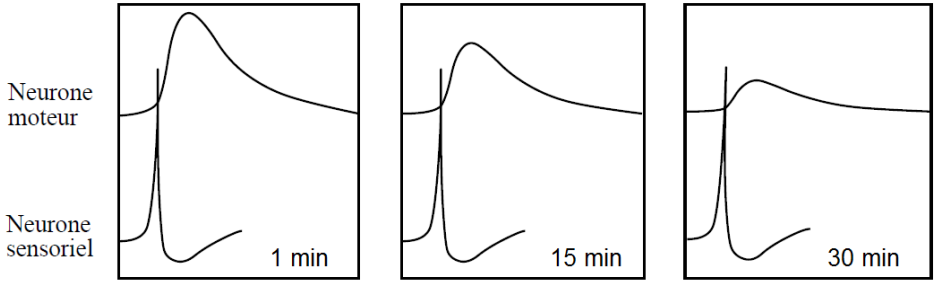


Figure 10. Habituation : lorsque la stimulation est répétée (quelques dizaines de fois), la réponse de l'animal devient de plus en plus faible, jusqu'à une absence de réaction au stimulus sensoriel. En bas à droite, le nombre de minutes après le début de l'expérience. A partir de 15 minutes, il n'y a plus de stimulations (d'habituation).

Sensibilisation

Si l'on répète la même expérience en créant après chaque stimulation du manteau un courant d'eau violent qui risque d'endommager les branchies, on observe alors l'effet inverse. Le courant d'eau sert de renforcement (fig. 11) et la réponse de l'animal au stimulus initial est augmentée (fig. 12). Cet effet est appelé sensibilisation.

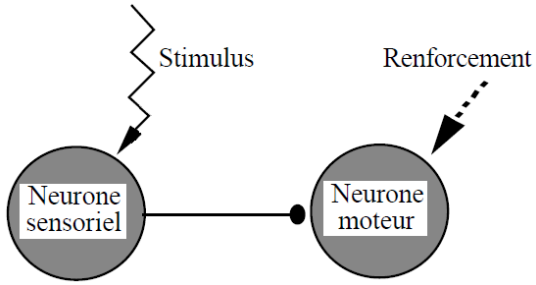


Figure 11. Circuit mis en jeu dans la sensibilisation.

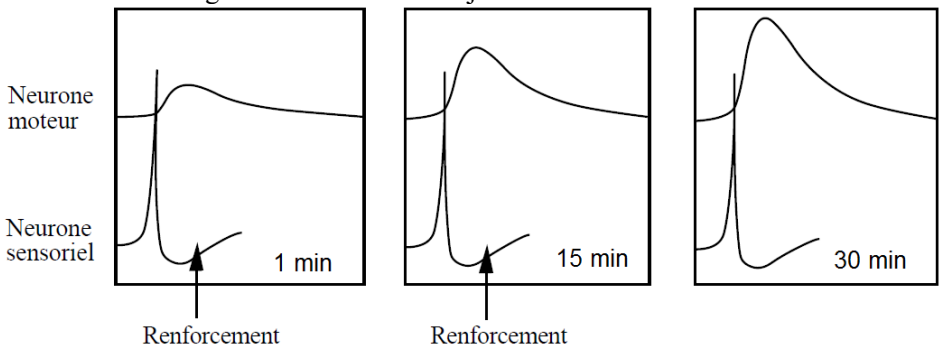


Figure 12. La sensibilisation : la réponse du neurone moteur au stimulus initial est augmentée par l'action du stimulus de renforcement. Le stimulus de renforcement n'est appliqué qu'après la réponse motrice.

Modification synaptique

Habituation et sensibilisation au niveau neuronal traduisent la fonction d'apprentissage au niveau de l'animal dans son ensemble. Il y a adaptation de la réponse à l'environnement. L'observation des synapses mises en jeu au microscope électronique montre des modifications physiques (fig. 13).

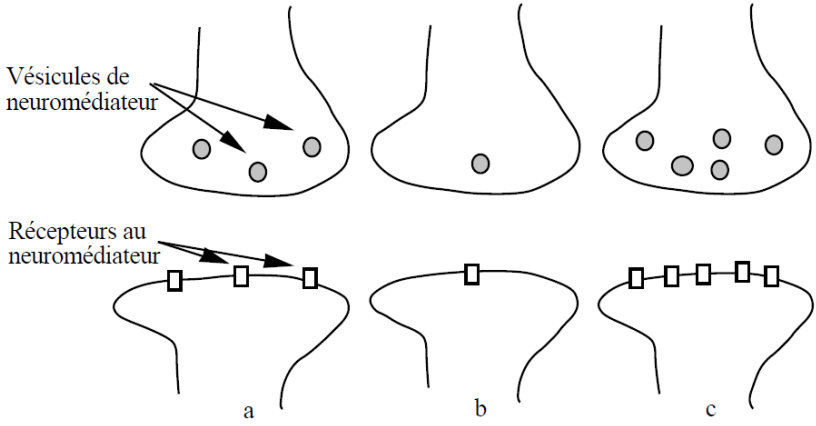


Figure 13. Modification physique de la synapse a) Témoin. b) Habituation : diminution du nombre de vésicules et du nombre de récepteurs. c) Sensibilisation : augmentation du nombre de vésicules et de récepteurs.

La vision et les étages de traitement

Nous avons vu des mécanismes de traitement de l'information au niveau de la coopération entre deux neurones. Il existe des structures plus complexes mettant en jeu des millions de neurones, qui rangés par étages de traitement diminuent la complexité de l'information, la rendant plus signifiante. C'est le cas du système visuel, sans doute le mieux étudié aujourd'hui.

Au niveau de la rétine, plusieurs dizaines de types différents de cellules codent les informations visuelles, chacune réalisant une fonction très spécialisée. Les images sont transformées en train d'impulsions nerveuses que le nerf optique véhicule vers le cerveau. Le cerveau élabore sa perception visuelle grâce à ces signaux. Cependant, au niveau de la rétine, il y a déjà traitement de l'information. En effet, on compte environ 150 millions de bâtonnets et 7 millions de cônes pour seulement 1 million de fibres au niveau du nerf optique.

On connaît aujourd'hui un certain nombre de circuits neuronaux de la rétine impliqués dans le traitement de l'information visuelle. Par exemple, à chaque cellule ganglionnaire correspond un champ récepteur : une zone précise du champ visuelle (disque d'un centimètre de diamètre à deux mètres de distance). Dès 1952, deux types de cellules ganglionnaires ont été répertoriés. En absence de stimulation lumineuse (obscurité), ces cellules émettent cependant spontanément un niveau

moyen de potentiels d'action. Les cellules à centre ON augmentent ce nombre d'impulsions lorsqu'un stimulus éclaire le centre du champ récepteur et deviennent silencieuses si le stimulus éclaire la périphérie du champ récepteur. Les cellules à centre OFF montrent un comportement inverse. La figure 14 montre un exemple d'architecture fonctionnelle pour une cellule ganglionnaire à centre ON. Cette opposition de fonctionnement entre le centre et la périphérie du champ récepteur permet d'améliorer les contrastes. On a découvert depuis d'autres cellules qui codent les directions de mouvements, etc.

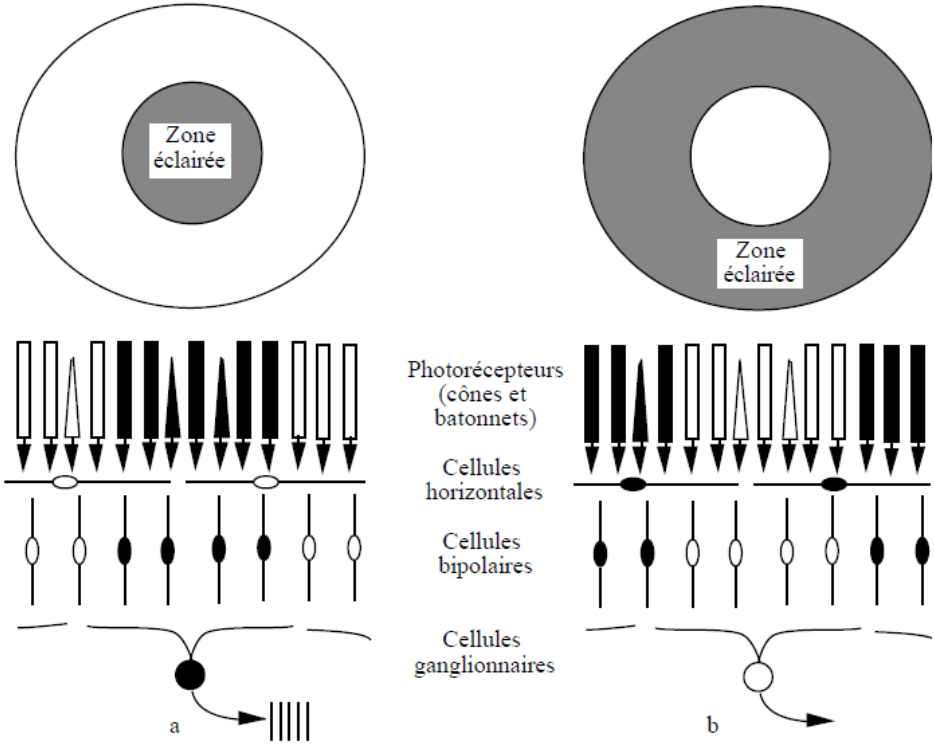


Figure 14. Traitement de l'information rétinienne par une cellule ganglionnaire à centre ON. En noir, les cellules actives. Les cellules horizontales ont une action inhibitrice sur les cellules bipolaires, elles s'opposent ainsi aux cellules photoréceptrices.

a) L'éclairage du centre du champ récepteur augmente le niveau d'activité.

b) L'éclairage de la périphérie du champ récepteur rend cette cellule silencieuse.

Au niveau du cortex visuel (arrivée du nerf optique), D. Hubel et H. Wiesel ont découvert l'existence de colonnes de dominance oculaire, spécifiquement excitées par un stimulus sous forme de barre dotée une orientation précise. La figure 15 montre une représentation schématique du cortex visuel.

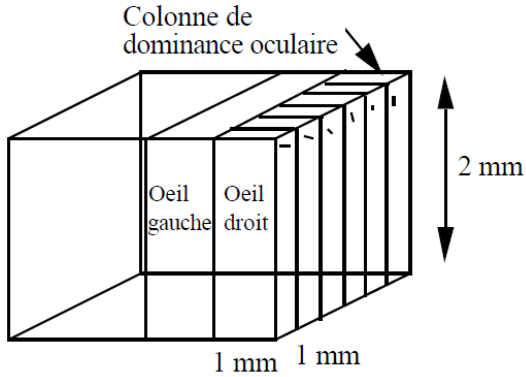


Figure 15. Représentation schématique du cortex visuel qui montre les colonnes de dominances oculaires et leur orientation privilégiée. On remarque l'alternance œil gauche - œil droit.

Nous avons vu une organisation topologique précise pour le traitement de l'information visuelle dont la construction semble génétique. Il existe néanmoins des possibilités d'apprentissage sur cette structure. Des expériences ont montré que l'élevage d'un chaton dans un univers composé uniquement de verticales va modifier ses perceptions jusqu'à le rendre pratiquement aveugle aux autres directions (horizontales et obliques) à l'âge adulte. L'étude histologique montre que la grande majorité de ses colonnes de dominances oculaires se sont "recyclées" dans les verticales.

Quels sont les mécanismes qui permettent de modifier le comportement des structures neuronales ? D. Hebb a proposé en 1949 une règle où la force de la connexion entre deux neurones augmente si il y a corrélation d'activité (si l'activation de l'une entraîne l'activation de l'autre). Cette hypothèse a depuis été complétée par J. P. Rauscheker et W. Singer qui proposent de modifier en les diminuant les forces des connexions non fonctionnelles (inutiles dans le contexte de fonctionnement actuel).

Remarquons que cette loi d'apprentissage initialement proposée pour les synapses excitatrices s'applique de la même manière pour les synapses inhibitrices. Effectivement, cela paraît contre productif puisqu'une connexion inhibitrice efficace (qui empêche l'activation du neurone aval) voit son efficacité diminuée du fait qu'elle est efficace... Une hypothèse récente[3] voit dans ce constat la cause même du sommeil : les ondes lentes du sommeil profond garantissent une activation locale de tous les neurones voisins et renforcent ainsi les connexions inhibitrices (qui ont perdu une partie de leur efficacité durant la période d'éveil).

Ce qu'il faut retenir :

Le neurone reçoit de l'information, et la transmet éventuellement. Il y a donc décision à son niveau. Cette décision est une action binaire qui transforme en « tout ou rien » une information qui était codée sur un plus grand nombre de valeurs (dans une première approximation, on peut estimer que le nombre de synapses d'un neurone donne le niveau de discrétisation de l'information reçue). Il s'agit d'une action violente, qui fait disparaître de nombreux « détails », mais qui régénère aussi l'information. Celle-ci devenant plus concise, devient aussi plus « pertinente ». La « décision » neuronale est donc indispensable à une extraction pertinente au sein d'un vaste volume de données (comme celles fournies en permanence par nos sens).

3 C. Touzet, "Sleep: the hebbian reinforcement of the local inhibitory synapses", *Med. Hypotheses*, 85:359-364, 2015.

I. HISTORIQUE

2. Modélisation du neurone

Le premier chapitre a présenté (succinctement) le neurone biologique. Nous allons maintenant le modéliser – c'est à dire le « simplifier » tout en conservant les propriétés qui nous semblent importantes – dans l'objectif d'obtenir au final un traitement de l'information « intelligent ».

Les RNA sont des modèles, à ce titre ils peuvent être décrit par leurs composants, leurs variables descriptives et les interactions des composants.

Structure

La figure 16 montre la structure d'un neurone artificiel. Chaque neurone artificiel est un processeur élémentaire. Il reçoit un nombre variable d'entrées en provenance de neurones amonts. A chacune de ces entrées est associée un poids *w* abréviation de weight (« poids » en anglais) représentatif de la force de la connexion. Chaque processeur élémentaire est doté d'une sortie unique, qui se ramifie ensuite pour alimenter un nombre variable de neurones avals. A chaque connexion est associée un poids. La figure 17 donne les notations que nous utilisons dans cet ouvrage.

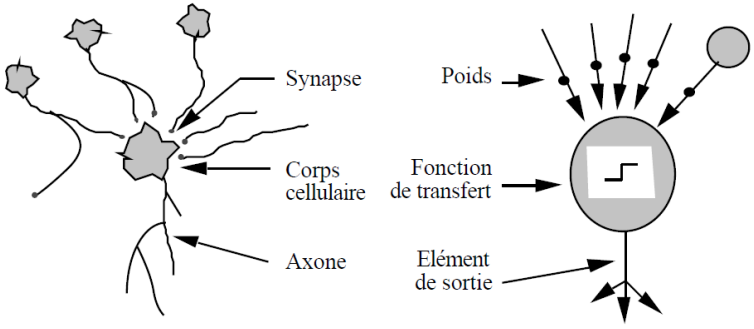


Figure 16. Mise en correspondance neurone biologique / neurone artificiel.

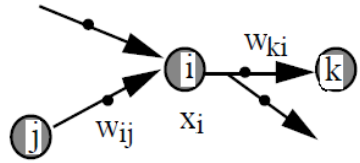


Figure 17. Pour le neurone d'indice *i*, les entrées sur celui-ci sont de poids w_{ij} alors que les connexions avals sont de poids w_{ki}.

Comportement

On distingue deux phases. La première est habituellement le calcul de la somme pondérée des entrées (*a*) selon l'expression suivante : $a = \Sigma (w_i * e_i)$.

A partir de cette valeur, une fonction de transfert calcule la valeur de l'état du neurone. C'est cette valeur qui sera transmise aux neurones avals. Il existe de nombreuses formes possibles pour la fonction de transfert. Les plus courantes sont présentées sur la figure 18. On remarquera qu'à la différence des neurones biologiques dont l'état est binaire, la plupart des fonctions de transfert sont continues, offrant une infinité de valeurs possibles comprises dans l'intervalle [0, +1] (ou [-1, +1]).

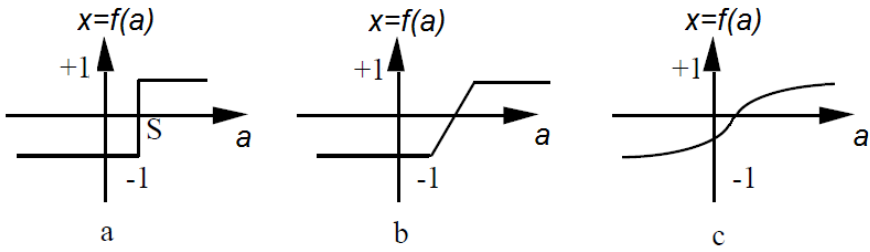


Figure 18. Différents types de fonction de transfert.
a) fonction à seuil (S , la valeur du seuil), b) linéaire par morceaux, c) sigmoïde.

Nous constatons que les équations décrivant le comportement des neurones artificiels n'introduisent pas la notion de temps. En effet, et c'est le cas pour la plupart des modèles actuels de réseaux de neurones,

nous avons affaire à des modèles à temps discret, synchrone, dont le comportement des composants ne varie pas dans le temps.

Variables descriptives
Ces variables décrivent l'état du système. Dans le cas des réseaux de neurones qui sont des systèmes non autonomes, un sous-ensemble des variables descriptives est constitué par les variables d'entrée, variables dont la valeur est déterminée extérieurement au modèle.

Structure d'interconnexion
Les connexions entre les neurones qui composent le réseau décrivent la topologie du modèle. Elle peut être quelconque, mais le plus souvent il est possible de distinguer une certaine régularité.

Réseau multicouche (au singulier) : les neurones sont arrangés par couche. Il n'y a pas de connexion entre neurones d'une même couche et les connexions ne se font qu'avec les neurones des couches avales (fig. 19). Habituellement, chaque neurone d'une couche est connecté à tous les neurones de la couche suivante et celle-ci seulement. Ceci nous permet d'introduire la notion de sens de parcours de l'information (de l'activation) au sein d'un réseau et donc définir les concepts de neurone d'entrée, neurone de sortie. Par extension, on appelle couche d'entrée l'ensemble des neurones d'entrée, couche de sortie l'ensemble des neurones de sortie. Les couches intermédiaires n'ayant aucun contact avec l'extérieur sont appelés couches cachées.

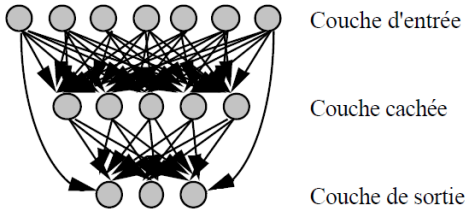


Figure 19. Définition des couches d'un réseau multicouche.

Réseau à connexions locales : Il s'agit d'une structure multicouche, mais qui à l'image de la rétine, conserve une certaine topologie. Chaque neurone entretien des relations avec un nombre réduit et localisé de

neurones de la couche avale (fig. 20). Les connexions sont donc moins nombreuses que dans le cas d'un réseau multicouche classique.

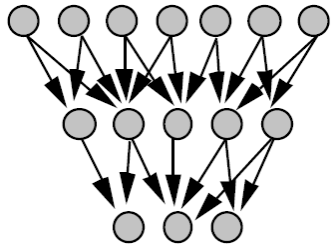


Figure 20. Réseau à connexions locales

Réseau à connexions récurrentes : les connexions récurrentes ramènent l'information en arrière par rapport au sens de propagation défini dans un réseau multicouche. Ces connexions sont le plus souvent locales (fig. 21).

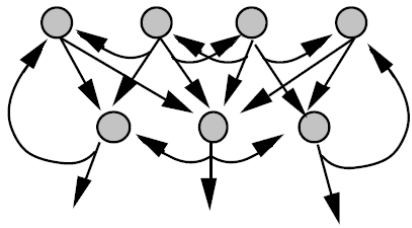


Figure 21. Réseau à connexions récurrentes

Réseau à connexion complète : c'est la structure d'interconnexion la plus générale (fig. 22). Chaque neurone est connecté à tous les neurones du réseau (et à lui-même).

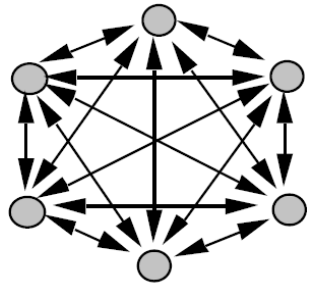


Figure 22. Réseau à connexion complète.

Il existe de nombreuse autres topologies possibles, mais elles n'ont pas eu à ce jour la notoriété des quelques unes que nous avons décrites ici.

Le Perceptron

Avant d'aborder le comportement collectif d'un ensemble de neurones, nous allons présenter le Perceptron (un seul neurone) en phase d'utilisation. L'apprentissage ayant été réalisé, les poids sont fixes. Le neurone de la figure 23 réalise une simple somme pondérée de ses entrées, compare une valeur de seuil, et fournit une réponse binaire en sortie. Par exemple, on peut interpréter sa décision comme classe A si la valeur de x est +1, et classe B si la valeur de x est -1.

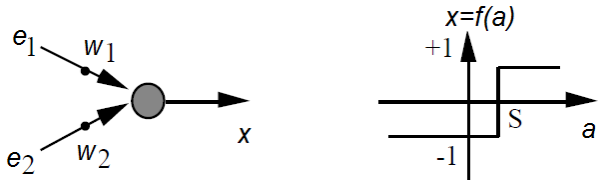


Figure 23. Le Perceptron : structure et comportement. Les connexions des deux entrées e_1 et e_2 au neurone sont pondérées par les poids w_1 et w_2 . La valeur de sortie du neurone est notée x. Elle est obtenue après somme pondérée des entrées (a) et comparaison à une valeur de seuil S.

Question : Sachant que les poids du Perceptron à deux entrées sont les suivants : $w_1 = 0.5$, $w_2 = 0.7$ et que la valeur de seuil est $S = 1.0$, déterminez son comportement, sachant que les comportements du ET logique, OU logique et OU exclusif sont rappelés ci-dessous.

ET		
e_1	e_2	x
1	1	1
1	-1	-1
-1	1	-1
-1	-1	-1

OU		
e_1	e_2	x
1	1	1
1	-1	1
-1	1	1
-1	-1	-1

OU exclusif		
e_1	e_2	x
1	1	1
1	-1	-1
-1	1	-1
-1	-1	1

Réponse : ET

Réseau multicouche en phase d'association

Le comportement collectif d'un ensemble de neurones permet l'émergence de fonctions d'ordre supérieure par rapport à la fonction élémentaire du neurone. Imaginer de prime abord un tel comportement n'est pas facile.

Soit un réseau multicouche composé de 361 (19 x 19) neurones d'entrée, 25 neurones cachés et 361 neurones de sortie. Ce réseau est une mémoire homo-associative (par opposition à hétéro-associative) qui donc associe à la lettre "a" présentée en entrée la même lettre en sortie (fig. 24). Présentons au réseau cette lettre avec quelques erreurs : un certain nombre de pixels ont été inversé (de blanc à noir ou inversement). L'image est composée de 19 x 19 pixels, chacun de ces pixels est associé à un neurone de la couche d'entrée. Chacun des neurones de la couche cachée reçoit 361 connexions (une pour chaque neurone d'entrée) et envoie sa sortie à chacun des neurones de sortie. Dans notre exemple, il y a 2 x (361 x 25) = 18050 connexions (ou poids).

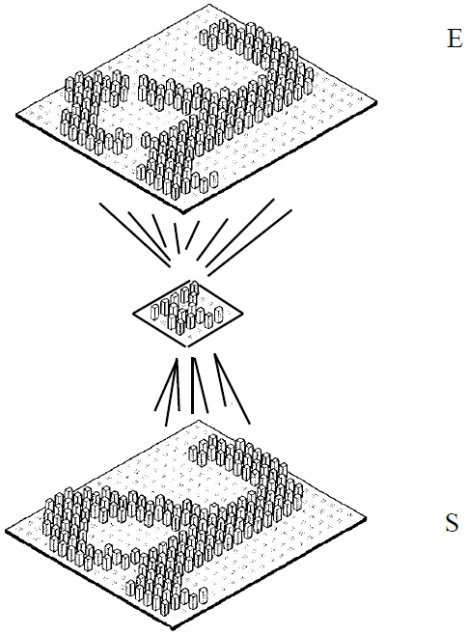


Figure 24. Comportement en phase de reconnaissance d'un réseau de neurone multicouche lors d'une tâche d'auto-association. Les neurones sont binaires. La valeur d'activation de chaque neurone est indiquée par la hauteur de la colonne. Les neurones sont rangés par couche, tous les neurones d'une couche sont connectés à tous les neurones de la couche suivante (aval).

La première étape est pour chaque neurone d'entrée de déterminer la valeur de son état selon la couleur du pixel correspondant (de l'image). Si les neurones qui composent le réseau sont binaires, on choisit arbitrairement de coder un pixel noir par un niveau d'activation du neurone égal à 1 (si le pixel est blanc alors le niveau d'activation du neurone correspondant est égal à 0).

La seconde étape est celle du calcul de la réponse qui se décompose en autant de sous-étapes qu'il y a de couches dans le réseau. Ainsi, chaque neurone de la couche d'entrée envoie sa valeur aux neurones de la couche cachée. Chacun des neurones de la couche cachées est en fait un Perceptron à 361 entrées. Chacun des neurones réalise la somme pondérée de ses entrées et compare à la valeur de seuil. Ce processus est effectué en parallèle et indépendamment pour tous les neurones de la couche cachée. Lorsque le vecteur d'activation de la couche cachée a été

obtenu, le même processus est répété avec les neurones de la couche de sortie. On considère ceux-ci comme 361 Perceptrons indépendants à 25 entrées.

La dernière étape est l'interprétation du vecteur d'activation de la couche de sortie par l'expérimentateur. Dans notre cas, on réalise l'opération inverse du codage initial, à savoir associer un pixel noir à chacun des neurones de la couche de sortie dont la valeur d'activation est égale à 1, un pixel blanc dans le cas contraire.

Dans la figure 24, il faut remarquer que si les vecteurs d'activation des couches d'entrée et de sortie semblent directement interprétables, il n'en est rien en ce qui concerne la couche cachée. Lorsque les neurones qui composent le réseau sont à valeur continue, les possibilités offertes sont plus nombreuses. L'image d'entrée peut être composée de plusieurs niveaux de gris. On associe alors arbitrairement à chaque niveau de gris un niveau d'activation du neurone spécifique. Le calcul du vecteur d'activation de la couche cachée reste identique dans son principe, nonobstant le fait que l'état de chaque neurone n'est plus binaire. L'interprétation de la réponse du réseau fournit alors une image composée de niveaux de gris.

Réseau à connexion complète

Chaque vecteur d'activation représente la réponse du réseau à une date particulière. Pour faciliter la représentation, nous avons déplié dans l'espace les évolutions temporelles du réseau à connexion complète (trois cycles – fig. 25). D'un instant au suivant, chaque neurone recalcule indépendamment et en parallèle son état. Rappelons que chacun des neurones est connecté à tous les autres, ce qui implique pour chacun d'entre eux de recevoir 361 connexions et d'envoyer sa sortie à ses 361 voisins. La principale différence entre les évolutions temporelles d'un réseau à connexion complète et le calcul de la sortie dans un réseau multicouche est que pour le premier les poids des connexions entre deux évolutions temporelles sont identiques, alors que pour le second, d'une couche à l'autre les poids des connexions sont différents.

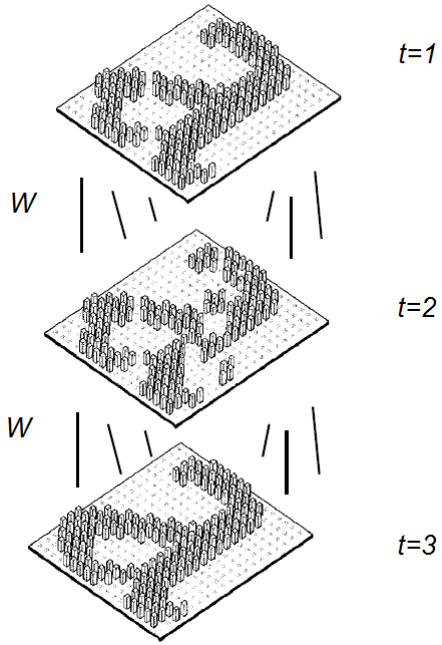


Figure 25. Évolution du vecteur d'activation d'un réseau à connexion complète sur une période de trois cycles. La matrice des poids *W* est complète (361 x 361 = 130321 poids). Entre deux "couches", c'est la même matrice de poids.

Réseau à inhibition latérale récurrente.

Les poids sont fixés *a priori* lors de la construction du réseau, il n'y a pas de phase d'apprentissage. La structure du réseau est représentée sur la figure 26, son comportement sur la figure 27. Il y a une seule couche de neurones. Les connexions sont localisées, chaque pixel d'entrée est en relation (excitatrice) avec un nombre réduit de neurones. De plus, on remarque la présence de connexions récurrentes inhibitrices localisées autour de chaque neurone.

En utilisation, une image est présentée en entrée. A la différence d'un réseau multicouche classique, la réponse n'est obtenue qu'après stabilisation de l'état de sortie. Le régime transitoire est dû au retour d'information depuis les neurones voisins. De fait, cette boucle doit être réalisée un certain nombre de fois avant que l'on obtienne une valeur fixe en sortie.

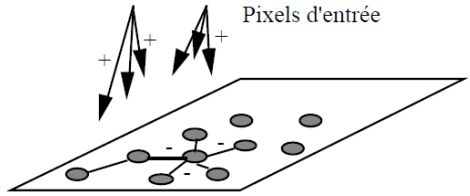


Figure 27. Architecture d'un réseau à inhibition latérale récurrente.
Chaque pixel d'entrée est connecté à un ensemble de neurones par des connexions excitatrices. Sur le réseau, chaque neurone réalise des connexions locales inhibitrices.

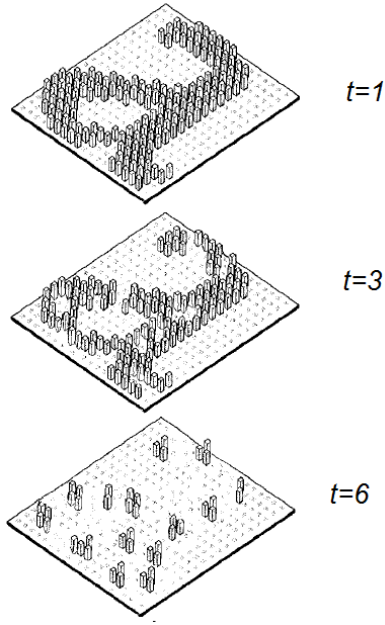


Figure 27. Comportement d'un réseau à inhibition latérale récurrente.
Seuls les pixels noirs de la forme d'entrée envoient une information sur le réseau. Pour chaque neurone, il y a compétition entre l'excitation en provenance de l'entrée et l'inhibition en provenance de ses voisins. On observe alors le développement au cours du temps de sous-groupes d'activité significatifs (angles, intersections, extrémités, etc).

Conclusion

Grâce aux quelques exemples de comportements vus, il est facile de comprendre que la disparition d'un ou même de plusieurs neurones (ou

de connexions) ne provoque pas une rupture brutale du traitement. En fait, la dégradation du comportement est fonction de la quantité d'éléments détruits. Cette propriété est désignée sous le terme de résistance aux pannes.

Par rapport aux données biologiques recensées au chapitre précédent, nous constatons :

- une réduction du nombre de connexions par neurone (de 10 000 à quelques centaines maximum),
- une réduction drastique du nombre de neurones pour un réseau artificiel (quelques centaines à comparer aux cent milliards du cerveau),
- une diminution de la complexité de la synapse et l'atypie des topologies proposées.

Ce qu'il faut retenir :

Les RNA doivent disposer de « nombreux » neurones, « nombreuses » synapses plastiques, et les neurones doivent prendre des « décisions ». Ceci est nécessaire et suffisant pour doter les RNA de capacités d'apprentissage et de généralisation – que nous aurons tendance à confondre (par anthropomorphisation) avec de l'intelligence.

3. Apprentissage et Perceptron

Définition
« L'apprentissage est une phase du développement d'un réseau de neurones durant laquelle le comportement du réseau est modifié jusqu'à l'obtention du comportement désiré. »

Les variables modifiées pendant l'apprentissage sont les poids des connexions. L'apprentissage est la modification des poids du réseau dans l'optique d'accorder la réponse du réseau aux exemples et à l'expérience. Il est souvent impossible de décider *a priori* des valeurs des poids des connexions d'un réseau pour une application donnée. A l'issu de l'apprentissage, les poids sont fixés : c'est alors la phase d'utilisation.

Certains modèles de réseaux sont (improprement) dénommés à apprentissage permanent. Dans ce cas, il est exact que l'apprentissage ne s'arrête jamais – cependant on peut toujours distinguer une phase d'apprentissage (en fait la mise à jour du comportement) et une phase d'utilisation. Cette technique permet de conserver au réseau un comportement adapté malgré les fluctuations dans les données d'entrées.

Au niveau des algorithmes d'apprentissage, il y a deux grandes classes selon que l'apprentissage est dit « supervisé » ou « non supervisé ». Cette distinction repose sur la forme des exemples d'apprentissage.

Dans le cas de l'apprentissage supervisé, les exemples sont des couples (Entrée, Sortie associée) alors que l'on ne dispose que des valeurs (Entrée) pour l'apprentissage non supervisé. Remarquons cependant que les modèles à apprentissage non supervisé nécessitent avant la phase d'utilisation une étape de labellisation effectuée par l'opérateur humain. Cette étape n'est pas autre chose qu'une « supervision ».

L'apprentissage par renforcement est entre les deux. Ses exemples d'apprentissage fournissent une qualification de l'erreur (« juste »,

« faux », « sans avis »), et non une quantification (comme pour le supervisé).

La loi de Hebb

La loi de Hebb (1949) est un exemple d'apprentissage non supervisé . Elle s'exprime de la manière suivante :

« Si 2 cellules sont activées en même temps alors la force de la connexion augmente ».

La modification de poids dépend de la co-activation des neurones pré et post synaptiques. La loi de Hebb (3ème colonne de la table suivante) qui explique le renforcement, a été étendue pour expliquer l'habituation (4ème colonne) :

x_i	x_j	∂w_{ij}	∂w_{ij}
0	0	0	0
0	1	0	-
1	0	0	-
1	1	+	+

∂w_{ij} est la dérivée partielle du poids, c'est à dire la modification qui sera appliquée à la valeur actuelle du poids ($w_{ij} = w_{ij} + \partial w_{ij}$). La dérivée partielle peut être calculée comme $\partial w_{ij} = x_i * x_j$ (co-activité des valeurs d'activation des deux neurones).

Algorithme d'apprentissage

L'algorithme d'apprentissage modifie de façon itérative (petit à petit) les poids. Il y a n neurones d'entrées, un unique neurone de sortie. Les exemples d'apprentissage sont des vecteurs donnant toutes les valeurs d'entrée et la valeur de sortie (e_1, … , e_n , x_d) . L'apprentissage continue tant qu'il y a des erreurs, c'est à dire tant que la sortie calculée est différente de la sortie désirée (la sortie désirée est celle donnée dans les exemples de la base d'apprentissage). On appelle itération d'apprentissage, le passage de tous les exemples de la base d'apprentissage une fois dans l'algorithme.

1/ Initialisation des poids w et du seuil S à des valeurs (petites) choisies au hasard.
2/ Présentation d'une entrée $E_p = (e_1, \ldots e_n)$ de la base d'apprentissage.
3/ Calcul de la sortie obtenue x pour cette entrée :

$$a = \Sigma (w_i * e_i) - S$$

(la valeur de seuil est introduite ici dans le calcul de la somme pondérée)

x = signe (a) (si $a > 0$ alors $x = +1$ sinon $a \leq 0$ alors $x = -1$)

4/ Si la sortie x est différente de la sortie désirée x_d pour cet exemple d'entrée E_p alors il y a modification des poids :

$$w_i = w_i + \alpha * (e_i * x_d)$$

α est une constante positive, qui spécifie le pas de modification des poids ($0<\alpha<1$).
5/ Tant que tous les exemples de la base d'apprentissage ne sont pas traités correctement (*i.e.,* qu'il y a modification des poids), retour à l'étape 2.

Exemple d'application de l'algorithme d'apprentissage de Hebb :
Réseau de 3 neurones : 2 neurones d'entrée et 1 de sortie. On désire réalisé l'apprentissage par le réseau du comportement suivant. Il y a 4 exemples de comportement (numérotés de 1 à 4). Les valeurs des neurones sont binaires (-1 et +1).

e_1	e_2	x	n°
1	1	1	(1)
1	-1	1	(2)
-1	1	-1	(3)
-1	-1	-1	(4)

1/ Conditions initiales : $\alpha = +1$, les poids w et le seuil S sont nuls.
2/ Calculons la valeur de x pour l'exemple (1) :
3/ $a = w_1 * e_1 + w_2 * e_2 - S = 0.0 * 1 + 0.0 * 1 - 0.0 = 0$

$a \leq 0 \Rightarrow x = -1$

4/ La sortie obtenue est différente de la sortie désirée (apprentissage) :

$w_1 = w_1 + \alpha * (e_1 * x_d) = 0.0 + 1*1*1 = 1.0$

$w_2 = w_2 + \alpha * (e_2 * x_d) = 0.0 + 1*1*1 = 1.0$

5/ On passe à l'exemple suivant (2) :

3/ $a = 1*1 + 1*-1 - 0.0 = 0$

$a \leq 0 \quad => \quad x = -1$

4/ La sortie est fausse, il faut donc modifier les poids :

$w_1 = 1.0 + 1*1*1 = 2.0$

$w_2 = 1.0 + 1*1*-1 = 0.0$

L'exemple suivant (3) est correctement traité :

$a = -2$ et $x = -1$ (la sortie est bonne).

On passe directement, sans modification des poids à l'exemple (4).
Celui-ci aussi est correctement traité.
On revient alors au début de la base d'apprentissage : l'exemple (1).
Il est correctement traité, ainsi que le second (2).
L'algorithme d'apprentissage est alors terminé car tous les exemples d'apprentissage ont été passés en revue sans modification des poids.

Question :
Recherchez les valeurs de poids qui résolvent le problème pour un réseau composé de 4 neurones d'entrée et d'un neurone de sortie ($w_1 = w_2 = w_3 = w_4 = 0$, $S = -0.1$) et la base d'apprentissage :

e_1	e_2	e_3	e_4	x	n°
1	1	1	1	1	(1)
-1	-1	1	-1	-1	(2)
1	-1	-1	1	-1	(3)
1	-1	-1	-1	1	(4)

Réponse : $w_1 = 3.0$, $w_2 = 1.0$, $w_3 = 1.0$, $w_4 = -1.0$

Remarque :

Il existe une possibilité de calculer les valeurs des connexions à partir des exemples en un coup (sans utiliser l'algorithme itératif). Si l'on

initialise les poids à 0.0 et que l'on présente les exemples de la base d'apprentissage, la valeurs des poids à l'issue de l'apprentissage est : $w_{ij} = \Sigma_p (x_{ip} * x_{jp})$ où p est l'indice de l'exemple dans la base d'apprentissage.

Perceptron

La règle de Hebb ne s'applique pas dans certain cas, bien qu'une solution existe. Un autre algorithme d'apprentissage a donc été proposé par Franck Rosenblatt en 1959 à Cornell (USA). Le Percep-tron (« Perception » et « électronique ») est destiné à des application de reconnaissance, notamment visuelle. L'algorithme d'apprentissage du Perceptron est semblable à celui utilisé pour la loi de Hebb. La seule différence se situe au niveau de la modification des poids qui devient :

$$w_i = w_i + \alpha * (x_d - x) * e_i$$

($x_d - x$) est une estimation de l'erreur. Plus il y a d'erreur, plus grande est la modification du poids.

Exemple d'application de l'algorithme d'apprentissage du Perceptron :
Réseau de 2 neurones d'entrée et 1 neurone de sortie. La base d'exemples d'apprentissage est la suivante :

e_1	e_2	x_d	n°
1	1	1	(1)
1	-1	-1	(2)
-1	1	-1	(3)
-1	-1	-1	(4)

1/ Conditions initiales : $w_1 = -0.2$, $w_2 = -0.2$, $S = +0.0$, $\alpha = +0.1$

2/ $a(1) = -0.2 - 0.2 - 0.0 = -0.4$

3/ $x(1) = -1$ (la sortie désirée x_d (1) = +1, d'où modification des poids)

4/ $w_1 = -0.2 + 0.1 * (1 - (-1)) * (+1) = 0.0$

$w_2 = -0.2 + 0.1 * (1 - (-1)) * (+1) = +0.0$

2/ $a(2) = 0.0 + 0.0 - 0.0 = 0.0$

3/ $x(2) = -1 \rightarrow$ Ok

2/ $a(3) = 0.0 + 0.0 - 0.0 = +0.0$

3/ $x(3) = -1 \rightarrow$ Ok

2-3/ $a(4) = 0.0 \rightarrow$ Ok

2-3/ $a(1) = 0.0 \rightarrow$ Faux

4/ $w_1 = 0.0 + 0.1 * (1 + 1) * (+1) = +0.2$

$w_2 = 0.0 + 0.1 * (1 + 1) * (+1) = +0.2$

2-3/ $a(2) = 0.2 - 0.2 - 0.0 = 0.0 \rightarrow$ Ok

2-3/ $a(3) = -0.2 + 0.2 - 0.0 = 0.0 \rightarrow$ Ok

2-3/ $a(4) = -0.2 - 0.2 - 0.0 = -0.4 \rightarrow$ Ok

2-3/ $a(1) = 0.2 + 0.2 - 0.0 = +0.4 \rightarrow$ Ok

5/ Tous les exemples de la base ont été correctement traités, l'apprentissage est terminé.

Le Perceptron réalise une partition de son espace d'entrée en 2 classes (A et B) selon la valeur de sa sortie (+1 ou -1). La séparation de ces deux zones est effectuée par un hyperplan (fig. 28). L'équation de la droite séparatrice est : $w_1 * e_1 + w_2 * e_2 - S = 0$. Durant l'apprentissage, la droite séparatrice se déplace jusqu'à couper correctement le plan afin que les exemples d'apprentissage soient correctement positionnés (chacun du côté de sa classe).

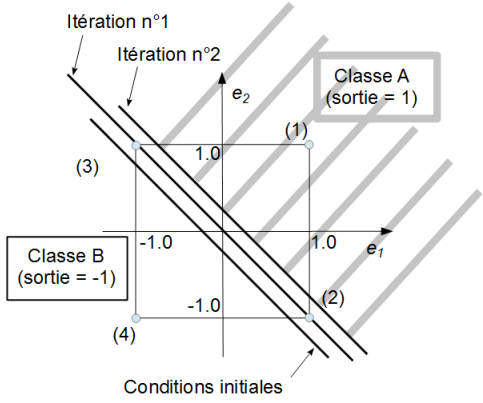


Figure 28. Partition de l'espace d'entrée de dimension 2 réalisée par un Perceptron se comportant comme un ET booléen. Les 4 exemples de la base d'apprentissage

sont les 4 arêtes du carré. Nous pouvons observer le déplacement de la frontière au cours de l'apprentissage. L'équation finale de la droite est : $0.2 * e_1 + 0.2 * e_2 - 0.0 = 0$

Remarque : Si les exemples d'apprentissage étaient différents, par exemple représentant le OU, alors c'est le comportement du OU qui aurait été appris avec le même algorithme d'apprentissage. D'autre part, il est possible de considéré que le seuil *S* est le poids d'une connexion dont le neurone amont est toujours dans l'état -1. La valeur de seuil se modifie alors en même temps que les autres poids. La convergence de l'algorithme vers la solution est plus facile (si cette solution existe), car nous venons d'ajouter un degré de liberté. Comme montré à la figure 29, cela ajoute la rotation à la translation de la frontière.

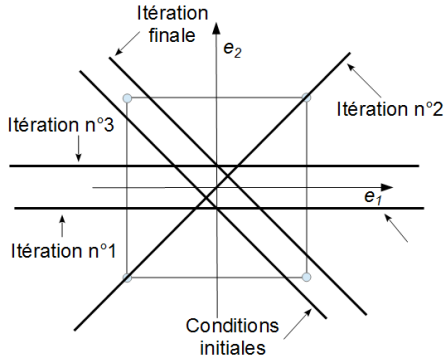


Figure 29. Conditions initiales identiques à la fig. 28, mais ici le seuil *S* devient variable.

Les différentes équations de droite frontière sont :

$-0.2 * e_1 + -0.2 * e_2 - 0.0 = 0$ (conditions initiales)

$0.0 * e_1 + 0.0 * e_2 - 0.2 = 0$ (itération n°1)

$0.2 * e_1 - 0.2 * e_2 - 0.0 = 0$ (itération n°2)

$0.0 * e_1 + 0.0 * e_2 + 0.2 = 0$ (itération n°3)

$0.2 * e_1 + 0.2 * e_2 - 0.0 = 0$ (itération finale)

Perceptrons et reconnaissance de chiffres manuscrits

Avec l'objectif de reconnaître des chiffres manuscrits (de 0 et 9), nous

allons utiliser 10 Perceptrons (1 pour chaque chiffre). Les chiffres ont été numérisés à l'aide d'un scanner (fig. 30).

Comme nous pouvons le constater, les différents chiffres n'ont pas la même taille, et donc le nombre de pixels est variable d'un chiffre à l'autre. Comme les entrées d'un RNA sont fixes, nous allons pré-traiter ces exemples en construisant une représentation fixe de chacun. Il s'agit d'une matrice de 3 x 5, et nous comptons le nombre de pixels noirs dans chaque case de cette matrice. Chaque exemple est alors un vecteur de dimension 15, et chaque composante a une valeur entre 0 et 80 environ (cf. fig. 31).

Figure 30. Quelques uns des chiffres manuscrits dans leur version scannée (la description complète des 190 chiffres est fournie en annexe).

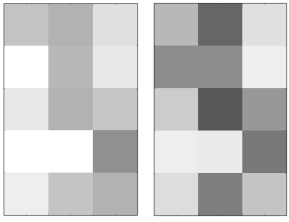


Figure 31. Pré-traitement des images numérisées, ici les deux premiers « 3 » de la fig. 30.

Nous disposons 19 exemples de chacun des neuf chiffres, soit 190

exemples en tout. Nous allons couper en deux cette base d'exemples afin de constituer une base d'apprentissage et une base de test. Les exemples de la base de test ne font donc PAS partie de l'apprentissage – et n'ont donc jamais été « vus » par le RNA.

Cette base de test nous permettra d'évaluer la généralisation, c'est à dire la capacité du RNA à répondre correctement lors de situations nouvelles (ici de nouvelles images de chiffres).

Normalement, la sélection des exemples d'apprentissage est un travail méticuleux car nous voulons nous assurer que cette base est bien représentative de ce que seront les tests. Pour l'instant, nous ne nous occupons pas de cette question, et sélectionnons les 100 premiers exemples pour l'apprentissage, les 90 restants pour le test.

99% de réussite sur la base d'apprentissage,70% sur la base de test.

Programme Python (10 Perceptrons)

```
# -*- coding: utf-8 -*-
import random
from BA import * # où se trouve la base d'apprentissage (cf. Annexe)

nb_perceptron = 10; nb_entree = 15; alpha = 0.5
W = []
for k in range (nb_perceptron) : # initialisation des poids [-1 et +1]
    W.append([])
    for i in range (nb_entree+1) :
       W[k].append(random.random()*2.0-1.0)
erreur = 1
iteration = 0
nb_ex_app = 10 # De 0 à nb_ex_app : la base d'apprentissage
while (erreur > 0) & (iteration < 1000) :  #apprentissage
  erreur = 0; iteration = iteration +1
  for ligne in BA2 : # un chiffre après l'autre
    for jj in ligne : # tous les exemples de ce chiffre
      j=jj%10 # (0, 1, 2, ...)
      for k in range (nb_perceptron) : # 10 perceptrons
         somme=0.0
         for i in range (nb_entree) :
            somme = somme + W[k][i] * BA[jj][i]/100.0
         somme = somme + W[k][i+1] * -1. # entree seuil
         Yobt = 0
         if somme >= 0 : Yobt = 1 # Sortie obtenue
         if Yobt != BAD[j][k] :
            erreur = erreur +1
            for i in range (nb_entree) :
               W[k][i]=W[k][i]+alpha*(BAD[j][k]-somme)*BA[jj][i]/100.0
            W[k][i]=W[k][i+1]+alpha*(BAD[j][k]-somme)*-1 # seuil
print  iteration,  "iterations ; nb_ex_app = ",  nb_ex_app
```

```
nb_ex_app = 0 # on repart de 0 : test sur l'ensemble des 190 exemples
erreur=0 # on compte le nombre des erreurs
for jj in range (nb_ex_app, len(BA)) : # test
   j=jj%10
   faux = 0 # a priori pas d'erreur pour cet exemple
   for k in range (nb_perceptron) : # 10 perceptrons
      somme=0.0
      for i in range (nb_entree) :
          somme = somme + W[k][i] * BA[jj][i]/100.
      somme = somme + W[k][i+1] * -1. # entree seuil
      Yobt = 0
      if somme >= 0 : Yobt = 1 # Sortie obtenue
      if Yobt != BAD[j][k] :
         faux = 1
         if Yobt==1 :
           print jj, " : propose ",  k, " alors que c'est un ", j
         else :
           print jj, " : ne propose rien alors que c'est un ",  k
      erreur = erreur + faux
print "Nb erreurs : ", erreur, "/",(len(BA)-nb_ex_app)
```

Exemple de résultat

```
 1000 iterations ; nb_ex_app =  10
 13  : propose  7  alors que c'est un  3
 68  : propose  2  alors que c'est un  8
 101  : ne propose rien alors que c'est un  1
 122  : ne propose rien alors que c'est un  2
 122  : propose  6  alors que c'est un  2
 …
 182  : ne propose rien alors que c'est un  2
 186  : ne propose rien alors que c'est un  6
 Nb erreurs :  30 / 190
```

Lecture : 1000 itérations d'apprentissage ont été effectuées (le nombre maximum). Donc il doit y avoir encore des erreurs sur la base d'apprentissage (les exemples de 1 à 100). C'est effectivement le cas pour les exemples 13 et 68.

Pour le cas 122, le Perceptron n°2 aurait du être à +1 pour signifier que c'était un « 2 ». Pour ce même cas, le Perceptron n°6 était à 1 alors qu'il aurait du être à 0. Il a fait croire que c'était un « 6 » (ce qui est faux).

Le nombre d'erreurs est de 30 réponses fausses sur les 190 essais – soit 16%. Cependant, si l'on fait l'on regarde dans le détails, les exemples d'apprentissage sont correctement reconnus à 98% (2 erreurs seulement). Donc, l'apprentissage se passe bien. A l'inverse, sur les exemples de test (non vus), la performance est de seulement 69%. La généralisation est

donc mauvaise : personne n'acceptera une application qui fait faux une fois sur trois !

A chaque fois que l'on lance le programme, le résultat est différent du fait du générateur de nombres aléatoires utilisés pour l'initialisation des poids. Il faut donc s'astreindre à réaliser au minimum une dizaine de simulations pour avoir une bonne idée des véritables performances. Avec une seule simulation, nous pouvons avoir beaucoup ou très peu de chance.

Les mauvaises performances en généralisation peuvent avoir deux causes :

1. Il y a un biais dans la sélection des exemples de la base d'apprentissage – qui n'est pas représentative de la base de test.
2. La structure du RNA ne facilite pas de la généralisation.

Représentativité

La notion de la représentativité de la base d'apprentissage est centrale pour les performances. Il serait pratique que nous puissions voir dans l'espace à 15 dimensions où se positionnent chacun des 190 exemples afin de choisir les 100 de la base d'apprentissage de façon à ce qu'ils représentent les régions où il y a beaucoup d'exemples et aussi ajouter les « exceptions » (ceux qui sont « seuls »).

Le modèle de la carte auto-organisatrice (que nous verrons plus loin) est exactement ce qu'il nous faut. La carte réalise une projection d'un espace multi-dimensionnel sur un sous-espace à deux dimensions, la projection respecte à la fois la topologie et la distribution de l'espace d'entrée.

Dans le cas présent, une projection des 190 exemples sur la carte est montrée à la figure 32. Nous voyons qu'il n'y a pas qu'un unique type de « 3 », mais plusieurs (certains « 3 » sont entre le « 8 » et le « 9 », d'autres entre le « 2 » et le « 5 » et enfin – les plus nombreux – entre le « 4 » et le « 5 »). Si la base d'apprentissage ne contient pas des exemples de chacun de ces trois types, alors la généralisation sera pratiquement impossible. Disposer de cette visualisation simplifie énormément la sélection d'une base d'apprentissage représentative.

9	–	–	8	7	7	–	6	6	6
9	3	8	–	7	7	6	6	–	6
9	–	–	7	7	7	–	6	4	4
–	9	–	8	7	7	5	5	5	4
–	9	8	–	–	–	5	–	4	–
9	9	8	8	–	8	–	5	–	4
9	–	–	1	–	8	2	–	5	3
0	0	8	–	2	2	–	3	–	3
0	–	2	1	2	–	–	3	–	3
0	2	–	2	1	1	1	–	5	–

Figure 32. Représentation 2-D (carte) de la distribution des 190 exemples de chiffres manuscrits. Il n'y a que 100 positions sur la carte, ce qui signifie que plusieurs exemples peuvent correspondent à la même position. Ce qui est proche sur la carte et a priori proches dans l'espace d'origine (à 15 dimensions).

Capacité de généralisation

La capacité de généralisation dépend avant tout du « sur-codage » réalisé par le RNA. Le codage – et donc le sur-codage – est réalisé par les connexions. Plus il y a de connexions (de degrés de liberté), plus il y a de sur-codage. Il devient alors possible d'être plus « fin » et « précis ».

Mais le nombre de connexions n'est pas tout, il faut aussi que l'architecture permette de réaliser des associations complexes (non-linéaires). Nous avons vu que chaque Perceptron coupe l'espace d'entrée au moyen d'un hyperplan de dimension -1 (par rapport à la dimension de l'espace des entrées). Dans le cas de 2 entrées, le Perceptron coupe avec une droite. Dans le cas de 3 entrées, il va couper avec un plan. Dans le cas de 4 entrées, il va couper avec un espace de dimension 3, etc. Dans le cas de 15 entrées, il coupe avec un espace de dimension 14.

Si nous désirons avec une idée de ce qui se passe, nous pouvons travailler avec la projection 2-D. La figure 33 montre une façon (parmi une infinité) de partionner les exemples avec 10 droites (puisqu'il y a 10 Perceptrons).

Nous voyons que le résultats n'est pas optimal. Notamment pour le cas des chiffres « 3 », qui sont répartis au sein de plusieurs régions différentes. Si les Perceptrons permettent de partitionner l'espace, il nous manque le moyen de regrouper certaines partitions codant pour la même classe. La figure 34 illustre cette question.

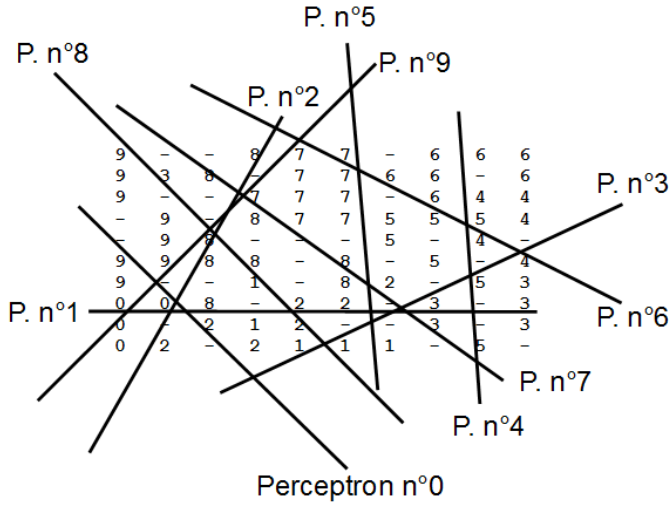


Figure 33. Partitionnement de l'espace d'entrée (et donc des exemples) avec 10 Perceptrons (c'est à dire 10 droites). La solution n'est pas optimale dès lors qu'une même classe occupe des régions différentes (fig. 34).

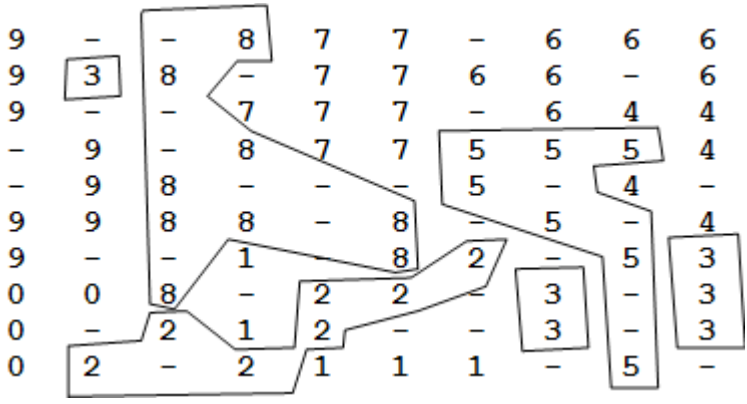


Figure 34. Il faudrait que les trois régions où se trouvent les « 3 » soient codées par la même classe. Découper l'espace ne permet pas d'y parvenir.

Ce problème a été identifié dès l'invention du Perceptron, et le plus petit cas où il se manifeste est lorsque l'on tente de faire apprendre le comportement du OU Exclusif. Il n'y a que 4 exemples d'apprentissage, mais deux d'entre eux (les n° 2 et 3) – bien qu'appartenant à la même classe –

sont dans deux régions séparées (figure 35). Il est impossible de résoudre le problème par un découpage de l'espace aussi précis soit-il.

e_1	e_2	x	n°
1	1	1	(1)
1	-1	-1	(2)
-1	1	-1	(3)
-1	-1	1	(4)

Figure 35. Le plus petit problème non-linaire et impossible à résoudre pour un Perceptron (ou une couche de Perceptrons). Les exemples pour lesquels à la réponse du RNA doit être « -1 » sont non contigus.

Par contre, il suffit de disposer d'une seconde couche de Perceptrons pour résoudre immédiatement le problème. Cette seconde couche réalise l'association de certaines régions (figure 36). Le besoin pour les Perceptrons multicouches s'est donc fait sentir très tôt, mais il aura fallu attendre 25 ans pour que la solution s'impose.

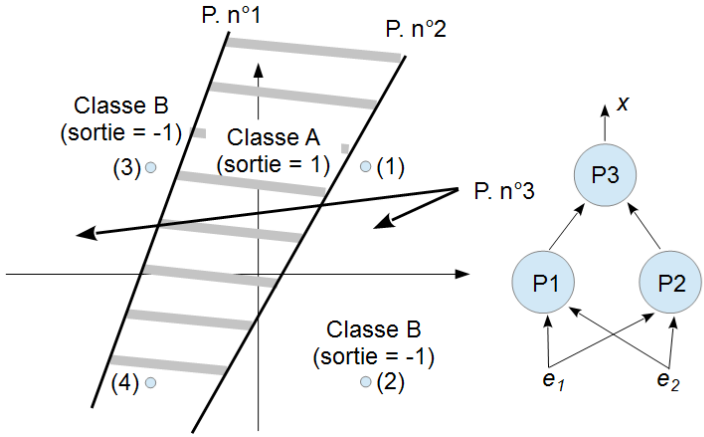


Figure 36. Résoudre le problème du OU Exclusif avec un Perceptron à deux couches : la première pour partitionner, la seconde pour associer.

Ce qu'il faut retenir :

Le Perceptron est le plus simple des RNA. Sa décision peut être vue comme une frontière partitionnant l'espace d'entrée. Son algorithme d'apprentissage est extrêmement simple à programmer et néanmoins très puissant – puisque comme nous l'avons vu il est capable d'apprendre à reconnaître des formes complexes (par exemple des chiffres manuscrits) avec une performance quasi optimale. Cependant, sa capacité de généralisation à des situations nouvelles (non vues durant l'apprentissage) est mauvaise du fait des limitations induites par sa structure à une seule couche. La représentativité de la base d'apprentissage est (elle-aussi) fondamentale.

4. Les réseaux de neurones multicouches (RMC)

Plusieurs couches de neurones permettent de réaliser des associations non linéaires entre l'entrée et la sortie. Les RMC sont ainsi capables de résoudre le cas du "ou exclusif" (cf. Perceptron). On sait depuis les années soixante que les possibilités de traitement des RMC sont supérieures à celle du Perceptron, cependant l'algorithme d'apprentissage manquait. Pour la couche de sortie, on peut appliquer l'apprentissage du Perceptron, mais comment modifier les poids pour les connexions plus internes ? Même une simple estimation de la valeur désirée pour chaque neurone de la couche cachée nous suffirait (puisque l'on pourrait alors estimer une erreur).

Il faudra attendre 25 ans (1985) pour que les limitations d'un apprentissage restreint à une seule couche de neurones (ou Perceptrons) soient levées. Cette découverte avait déjà eu lieu une quinzaine d'années plus tôt, mais l'époque était alors aux systèmes experts, aux systèmes à base de règles, à l'inférence et aux logiques plus ou moins floues. C'est donc une re-découverte faite indépendamment et simultanément en trois endroits différents de la planète – preuve s'il en est que cette fois les chercheurs n'allaient pas ignorer l'avancée. L'un des trois découvreurs est un Français – Yann LeCun – aujourd'hui à la tête du laboratoire de recherche de Facebook à Paris (après une brillante carrière aux Bell Labs notamment).

L'apprentissage pour les RMC a reçu le nom (barbare) d'algorithme d'apprentissage par rétro-propagation du gradient d'erreur quadratique – ce qui traduit exactement les faits : le gradient (l'équivalent d'une dérivée) de l'erreur élevée au carré (d'où le terme quadratique) est utilisée à rebours (rétro-propagation) pour permettre l'estimation de l'erreur réalisée par chaque neurone interne permettant ainsi la modification des poids.

Le principe utilisé par la rétropropagation ("backpropagation" en anglais) de gradient est la minimisation d'une fonction dépendante de l'erreur. Il s'agit d'une méthode générale, largement employée dans d'autres domaines tels que la Physique. Une perception intuitive de cet algorithme consiste à considérer celui-ci comme la recherche sur la surface de coût de la position de coût minimal. Pour un RMC, à chaque configuration de poids correspond une performance (le taux d'erreurs sur la base d'exemples d'apprentissage). Selon les valeurs des différents poids du RMC, l'erreur réalisée par celui-ci est plus ou moins grande. Lorsque l'on calcule l'erreur pour chaque configuration de poids, on obtient une surface hérissée de pics (erreurs importantes) et de creux (moins d'erreurs), c'est la « surface de coût ».

En pratique, il est impossible de calculer la performance du RMC pour chaque configuration de poids. S'il n'y a que 10 connexions dans le RMC, et que chaque poids est codé comme un entier dans l'intervalle (-10, 10), alors le nombre de configurations de poids différentes est de 10^{20} (ce qui est impossible à tester). Il s'agit pourtant d'un nombre de connexions très faible (habituellement plusieurs centaines sont impliquées). Comme le calcul exact de toutes les possibilités est impossible, on se sert de la propriété de continuité de la surface de coût (qui n'est pas chaotique) pour ne faire que quelques calculs – lesquels fournissent des estimations locales de la pente de la surface (les fameux « gradients »).

La minimisation du gradient permet de parcourir la surface de coût orthogonalement aux courbes de niveau. Les problèmes rencontrés durant l'apprentissage résultent des zones très plates (gradients pratiquement nuls) et des minima locaux (ce qui correspond à des solutions moins efficaces que celle proposée par le minimum global).

Par rapport à un Perceptron, les RMC nécessitent des neurones dont la fonction de transfert est continue (la « fonction seuil » du Perceptron est discontinue). L'erreur minimisée par l'algorithme de gradient est une erreur quadratique (c'est à dire « au carré ») car l'algorithme requiert une erreur qui va décroître avec l'amélioration des performances. On peut tout aussi bien choisir une erreur portée à la puissance 4, ou la valeur absolue de l'erreur[4].

4 mais pas une erreur qui peut être positive et aussi négative car comment interpréter une erreur de -3 par rapport à une erreur de +1. C'est plus grand ou plus petit ?

L'apprentissage par rétro-propagation du gradient d'erreur est très proche de celui du Perceptron. La seule différence est que le calcul de la modification des poids varie selon qu'il s'agit d'un neurone de la couche de sortie ou d'une couche cachée.

Algorithme de la rétro-propagation du gradient :

1/ Initialisation des poids à des valeurs aléatoires de faible grandeur

2/ Sélection d'un exemple d'apprentissage dans la base d'apprentissage

3/ Présentation de la forme d'entrée sur la couche d'entrée du réseau

4/ Calcul par propagation de la sortie obtenue

5/ Si erreur ($e_i = (d_i - x_i) \neq 0$) en sortie alors pour tous les neurones i (depuis la sortie jusqu'à l'entrée) :

- Si i est un neurone de sortie alors $y_i = 2e_i * f'(a_i)$

- Si i est un neurone caché (ou d'entrée) alors $y_i = f'(a_i) * \sum_k (w_{kj} * y_k)$

(k : neurones compris entre la couche actuelle et la couche de sortie)

6/ Application de la procédure de gradient :

$w_{ij}(t+1) = w_{ij}(t) + \alpha \, . \, y_i \, . \, x_j$

α est un gain fixé par l'utilisateur.

7/ Tant que l'erreur est trop importante, retour à l'étape 2 (on prend alors l'exemple suivant).

Cet algorithme, bien que très simple à implanter, nécessite un certain savoir-faire pour une utilisation efficace. En effet, la convergence n'est pas prouvée et de multiples variables sont à ajuster précisément en fonction du problème traité.

Parmi ces variables à fixer, citons par exemple : les paramètres apparaissant dans les différentes équations (le « pas » de modification des poids (α), la pente de la fonction sigmoïde (θ), etc.), la sélection des exemples pour l'apprentissage et le test, l'ordre de présentation et les distributions relatives des exemples dans la base d'apprentissage, le

choix du codage des informations en entrée et en sortie, la structure du réseau (présence éventuelle de connexions directes de la couche d'entrée sur la couche de sortie pour traiter à ce niveau la partie linéaire du problème, limitation pratique du nombre de couches, taille de la couche cachée), la configuration initiale des poids, le nombre d'itérations d'apprentissage, ...

NETtalk (en Français, « le réseau qui parle ») fut l'application développée par les découvreurs Californiens de la « rétro ». Cette application fit beaucoup pour la promotion de la « rétro ». Il s'agit pour le RMC d'apprendre à prononcer du texte en anglais (fig. 37). Un exemple d'apprentissage est constitué d'une chaîne de 7 caractères et de sa transcription phonétique du 4ème caractère. Les 3 caractères avant et aussi ceux d'après servent à identifier le contexte (un « i » ne se prononce pas de la même façon selon qu'il est précédé d'un « a » ou d'un « t », par exemple).

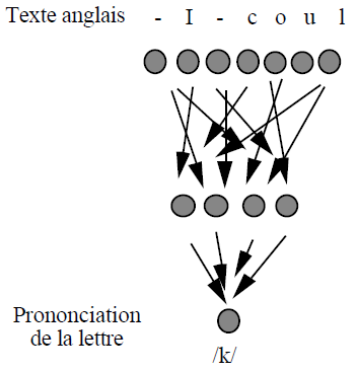


Figure 37. NETtalk : 309 neurones (3 couches) et 10629 poids (80 cellules cachées). Après 12 h d'apprentissage, la performance est de 95% sur le texte d'apprentissage (1000 mots) et 90% sur du texte nouveau (mots nouveaux). On a connecté en sortie un synthétiseur de paroles, le texte est compréhensible. De plus, il semble que durant l'apprentissage, NETtalk se comporte de la même manière qu'un enfant apprenant à lire (même type de fautes, ...).

Y. le Cun a proposé l'application des techniques connexionnistes à un problème de diagnostic médical en cas d'urgence. Dans ce cas, le réseau met en correspondance l'espace de départ constitué des symptômes avec

l'espace d'arrivée composé des diagnostics possibles. La fonction associant les symptômes avec les diagnostics est apprise par le réseau à partir d'un ensemble de cas réels.

Le terme "application" doit cependant être pris avec précaution. En effet, il faut distinguer entre les "applications candidates" qui sont en principe soluble par la technique connexionniste, les applications en cours de développement dont la faisabilité a été démontré sur un cas simplifié et les "applications prouvées", peu nombreuses. Réaliser une application, c'est d'abord exprimer le problème sous la forme d'une mise en correspondance de deux espaces, puis construire une base d'apprentissage représentative des données et enfin choisir, en se référant à son expérience, le modèle de réseau et ses paramètres. Il faut aussi préalablement définir les critères de mesure des performances (construction de la base de test), les prétraitements et le codage sur le réseau.

Exemple du diagnostic des douleurs abdominales

Expression du problème : Mise en correspondance de l'espace des symptômes avec celui des diagnostics à partir d'exemples de diagnostics réalisés par un expert humain.

Modèle : La base d'exemples est constituée de 5765 malades admis aux urgences. Chaque cas est décrit par 132 symptômes qualitatifs (par exemple le sexe) ou semi-quantitatifs (par exemple l'age), assortis à l'un des 22 diagnostics possibles. Environ 80% des symptômes seulement sont connus pour chacun des malades. On choisi de conserver 1740 cas pour le test (30%) tandis que 4027 cas (70%) sont utilisés pour l'apprentissage. A priori, la relation entre ces deux espaces est complexe (en particulier non linéaire) ce qui implique l'utilisation d'un réseau multicouche (ou de plusieurs réseaux multicouches). En fonction du codage utilisé pour représenter les entrées et les sorties, le réseau se compose d'une seule couche cachée de 62 neurones, de 316 entrées et 22 cellules de sortie. Il y a donc 3380 poids et 400 neurones. Les connexions intercouches ne sont pas complètes.

Performance : Après application de la procédure de rétro-propagation de gradient, les performances mesurées sur la base d'apprentissage sont de

75% et sur la base de test de 71% (seulement). Il faut relativiser ces performances en ce sens qu'un praticien en situation d'urgence ne fourni le bon diagnostic que dans 60% des cas. D'autre part, l'absence de certaines valeurs au niveau des symptômes posent problème. De même, il est possible que des incohérences existent au niveau de la base d'apprentissage ou de test (deux situations symptomatiques identiques avec un diagnostic différent par exemple). C'est là l'un des principaux avantages de l'approche connexionniste que de pouvoir rester efficace même dans ces situations.

Dérivation de la rétropropagation de gradient

Il faut trouver une configuration de poids qui minimise un critère d'erreur. On définit donc une fonction de coût :

$C(W) = M[C_p(W)] = M[\sum_j e_{pj}^2(W)]$ avec $e_{pj} = (d_{pj} - x_{pj})$

où j indique un numéro d'indice pour les neurones de sortie et p indique un exemple d'apprentissage. M est l'opérateur de moyennage, c'est une estimation de la moyenne temporelle dans le cas stochastique. On réalise donc la moyenne des erreurs obtenues pour chacun des exemples de la base d'apprentissage.

L'algorithme du gradient permet de trouver une configuration minimale pour la fonction de coût, il suffit d'appliquer de manière itérative la formule :

$W = W - \alpha * \partial C(W)$

où ∂C est la dérivée partielle de C par rapport à tous les w_{ij} .

Cet algorithme nécessite une fonction (f) continue, non-linéaire et différentiable comme fonction de transfert du neurone (θ est une constante positive qui donne la pente de la sigmoïde, par exemple 0,2).

$a_i = \sum_j w_{ij} * x_i$; $f(a_i) = (e^{\theta a_i} - 1) / (e^{\theta a_i} + 1)$

La linéarité de l'opérateur de moyennage permet d'effectuer tous les calculs sur les valeurs instantanées de la fonction de coût.

$\partial C/\partial w_{ij} = \partial C/\partial a_i * \partial a_i/\partial w_{ij}$ (dérivation des fonctions composées)

Sachant que $a_i = \sum_j w_{ij} * x_i$ alors $\partial C/\partial w_{ij} = \partial C/\partial a_i * x_j$

Il nous reste donc à calculer les $\partial C/\partial a_i$ que nous notons $- y_i$

Si i est une cellule de sortie : $y_i = -\partial C/\partial a_i = -\partial \sum_q e_q^2$

Après dérivation, $y_i = -\sum_q e_q^2 / \partial x_i * \partial x_i / \partial a_i$

($\partial e_q^2 / \partial x_i$ est nulle si $q \neq i$ pour toute cellule de sortie, indépendantes de fait).

d'où $y_i = -\partial e_q^2 / \partial x_i * \partial x_i / \partial a_i = -2e_i * \partial e_i / \partial x_i * \partial x_i / \partial a_i$

Sachant que pour les cellules de sortie $e_i = (d_i - x_i)$, alors $\partial e_i / \partial x_i = -1$

Résultat : $y_i = 2e_i * f'(a_i)$

Si i est une cellule cachée (ou d'entrée) : $y_i = -\partial C/\partial a_i$

Posant l'hypothèse que tous les y_k auxquelles la cellule i envoie sa sortie sont connus alors

$y_i = \sum_k (-\partial C/\partial a_k * \partial a_k/\partial a_i) = \sum_k (y_k * \partial a_k/\partial x_i * \partial x_i/\partial a_i)$

Soit $y_i = \sum_k (y_k * w_{kj} * f'(a_i)) = f'(a_i) * \sum_k (w_{kj} * y_k)$

Le calcul du gradient attaché à une cellule utilise les poids qui les relient aux cellules avals ainsi que des gradients correspondants y_k (fig. 38).

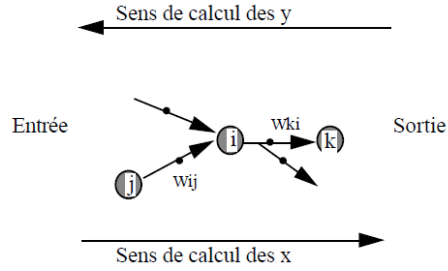


Figure 38. Définition d'un sens de propagation de l'information opposé pour le calcul des x et des y.

Ce qu'il faut retenir :

Deux couches de neurones permettent de positionner à la fois des frontières et des unions. Normalement, cela peut résoudre tous les problèmes (TSK), mais la réalité est autre car une chose est de « pouvoir », mais c'en est autre que de « savoir » le faire[5]. Nous verrons donc plus loin dans cet ouvrage, les réseaux de neurones profonds (une dizaine de couches).

5 En 1900, Hilbert conjectura que les équations algébriques de haut degré ne peuvent pas être résolues par des fonctions construites comme substitution de fonctions continues de deux variables. En 1957, Kolmogorov contredit cette hypothèse en présentant son théorème de superposition (TSK), qui permet d'écrire toute fonction multivariée comme sommes et compositions de fonctions monovariées. Toutefois, Kolmogorov ne proposa pas de méthode pour la construction de ces fonctions monodimensionnelles et démontra seulement leur existence.

5. Les cartes auto-organisatrices (CA)

La Théorie neuronale de la Cognition[6] (TnC) soutient que le cerveau ne traite pas l'information, mais la représente. Les représentations sont donc centrales. La première représentation découverte a été baptisée « homonculus ». Elle est due au neurochirurgien canadien Wilder Penfield qui dans les années 1950 profitait de ses opérations à crâne ouvert pour exciter, avec une petite électrode, le cortex en divers points. Il nota que dans une certaine région qu'on baptisa cortex sensoriel, la stimulation engendrait chez le sujet l'impression qu'il avait été touché sur une partie de son corps. Une stimulation corticale légèrement déplacée engendrait une impression émanent d'un point voisin (du corps). Il constata de plus que la surface corticale dévolue aux différentes parties du corps était directement proportionnelle à la densité des capteurs proprioceptifs de la peau (plus grande au niveau de la pulpe des doigts que dans le dos par exemple).

La figure 39 montre l'homoncule sensoriel, c'est à dire la correspondance entre les localisations corticale et proprioceptive. Par rapport à l'homoncule original présenté ici, les seules « erreurs » de Penfield ont trait à la localisation des organes génitaux que l'on sait aujourd'hui être – logiquement – placés en haut des cuisses (la pudibonderie de l'époque ayant rejeté ceux-ci le plus loin possible). Penfield a aussi cartographié un homoncule sensori-moteur, où l'excitation électrique à la surface du cortex sensori-moteur engendre une contraction musculaire involontaire.

6 Du même auteur, publiée aux éditions la Machotte en 2010 (tome 1),et 2014 (tome 2).

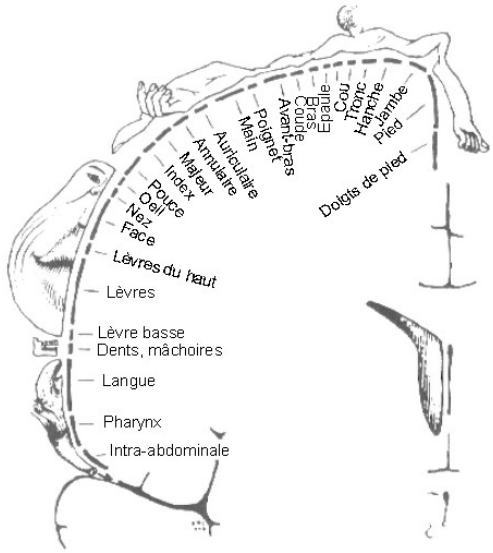


Figure 39. Homoncule sensoriel de Penfield. D'après W. Penfield, T. Rasmussen, *The cerebral cortex of man*, Macmillan, 1950, pp. 214-215.

L'homoncule est une carte, permettant de faire le lien entre une localisation au niveau du corps et une activité à la surface du cortex. Il y a bien d'autres cartes. En fait, l'ensemble du cortex est constitué de cartes. Le cortex se présente comme un tissu nerveux de 2 millimètres d'épaisseur sur 60 x 60 cm (fig. 40). Sur les 2 mm d'épaisseur se répartissent (chez l'homme) six couches de neurones, des petits, des gros, certains ont des connexions inhibitrices, d'autres excitatrices, etc. L'important est de savoir que, d'après nos connaissances actuelles, n'importe quelle localisation corticale est potentiellement en contact avec n'importe quelle autre – même si une distance de 60 cm représente une dimension considérable à l'échelle du neurone.

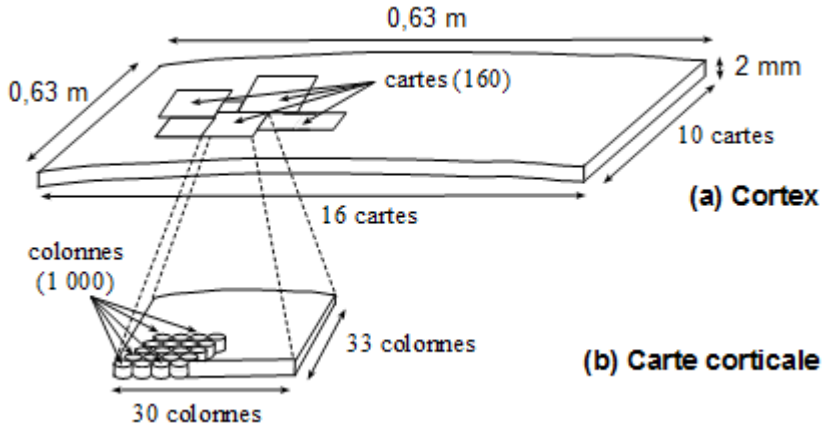


Figure 40. (a) Le cortex est organisé en cartes corticales (environ 160). (b) Chaque carte corticale contient plusieurs centaines de colonnes corticales, chacune comprenant un millier de micro-colonnes.

D'autres chercheurs, David Hubel et Torsten Wiesel, ont réalisé un travail considérable (qui leur valu le prix Nobel de Médecine 1981) en découvrant les « colonnes de dominance oculaire » dans les années 1960-1970, puis en décrivant (1970-1980) différents traitements réalisés par les neurones du cortex visuel (cf. fig. 15). Nous retenons de leurs efforts que les neurones corticaux sont assemblés en micro-colonnes de 110 neurones – elles-même rassemblées au sein d'unité plus importante les macro-colonnes (que l'usage désigne sous le seul terme de « colonne »). Chaque colonne traverse de part en part le cortex (les 2 mm d'épaisseur). Le nombre de neurones au sein d'une colonne est estimé à 110 000 chez l'homme. La structure du cortex est relativement homogène, chaque neurone appartient donc à une colonne corticale. Les colonnes corticales voisines codent pour des informations similaires. Les informations visuelles en provenance de l'oeil alimentent les cartes spécifiques de la vision, lesquelles cartes forment le cortex visuelle ; les informations auditives sont traitées par le cortex auditif, etc.

La TnC fait de la colonne corticale l'unité de représentation de l'information pour la cognition. Nos programmes informatiques vont donc simuler des colonnes et non pas des neurones. Ce choix est justifié par plusieurs considérations. Les neurones d'une même colonne corticale répondent spécifiquement pour le même stimulus – ils sont donc « inter-

changeables ». Ceci se comprend dès lors qu'on se rappelle qu'un neurone biologique dispose de réserves énergétiques limitées. Il ne peut pas répondre (à sa fréquence d'excitation maximale) plus de quelques dixièmes de seconde. Pourtant nous savons qu'une idée (par exemple) va rester présente à notre « conscience » pendant au minimum quelques secondes. Elle sera donc être codée par l'activation successives de dizaines, centaines, voire milliers de neurones. Dans notre modélisation, la colonne corticale est infatigable. De plus, elle permet de coder des niveaux d'activation – tandis que le neurone n'offre qu'un codage binaire (excité ou non excité). Remarquons que le nombre de colonnes d'un cerveau humain est de seulement 5 millions, beaucoup plus en phase avec les puissances de calcul aujourd'hui disponibles.

Dans les années 1970, Teuvo Kohonen, un chercheur finlandais, modélise la carte corticale. Ce modèle passera à la postérité sous les noms de carte topologique, de carte auto-organisatrice ou simplement de carte de Kohonen. C'est un réseau de neurones artificiels de type compétition : toutes les colonnes de la carte entre en compétition pour chaque entrée soumise, et c'est la plus rapide à s'activer qui gagne. La gagnante inhibe ses voisines, qui ne pourront donc pas s'activer pour cette entrée. On dit que la plus rapide est celle qui « représente » (le mieux) les données en entrée. Dans le cas du modèle informatique, et parce que la gestion du temps est délicate, un substitut mathématique remplace la vitesse d'activation. Il s'agit de la distance entre les entrées du réseau et le vecteur des poids synaptiques de la colonne. Ceci nécessite quelques explications. Tout d'abord, c'est la première fois que le terme poids synaptique apparaît.

Le poids synaptique désigne la force de la connexion entre deux neurones. Plus ce poids est grand, plus la connexion est efficace – ce qui signifie que l'excitation du neurone amont a de grandes chances d'exciter le neurone aval. S'il est négatif, alors la connexion est inhibitrice. L'activation du neurone amont aura pour effet de réduire la probabilité que le neurone aval passe dans un état excité. Ce que nous désignons par « poids synaptiques » (avec un « s ») est l'ensemble des forces de (toutes les) connexions du neurone. Dans le cas de la carte auto-organisatrice, comme il n'y a pas de neurone, mais des colonnes, il s'agit de l'ensemble des forces de connexion entre les neurones codant l'information en entrée (de la carte) et une colonne.

Les poids synaptiques sont parmi les variables d'un réseau de neurones biologiques. La valeur de poids peut changer en quelques secondes, et ce changement devient permanent – jusqu'au prochain changement. La mémoire (biologique) est réalisée par l'ajustement des valeurs des poids. Les autres variables (nombre de neurones, ajout d'un nouveau poids) évoluent beaucoup plus lentement. Il faut des jours pour qu'une nouvelle connexion s'établisse entre deux neurones qui n'en avaient pas. Il faut des semaines pour qu'un neurone biologique soit créé et devienne opérationnel. La carte de Kohonen est une modélisation – c'est-à-dire une simplification de la réalité. Elle utilise un unique type de variables : le poids des connexions entre entrées et colonnes.

La carte de Kohonen réalise une projection des données d'entrées appartenant à un multi-dimensionnel espace (de grande dimension) sur un sous-espace de dimension 2. Ce sous-espace est celui de la carte. Ce n'est pas un exactement un plan, puisque celui-ci devrait être « plat ». La carte se « courbe » dans l'espace multi-dimensionnel de manière à minimiser la distance de projection des données vers la carte. Ce faisant, elle permet de réduire les erreurs dues à la projection. Ainsi, les données voisines dans l'espace multidimensionnel sont représentées sur la carte par des colonnes voisines. Cette propriété est loin d'être facile à obtenir. Les Statistiques, qui forment une discipline où la réduction de dimension est un outil très employé, bénéficient de l'inspiration neuromimétique pour leurs meilleurs outils. C'est ainsi que l'analyse en composantes curvilignes est une extension de la carte de Kohonen.

La figure 41 montre une carte auto-organisatrice en train d'apprendre à représenter une distribution de données d'un espace de dimension 2. C'est un exemple pédagogique, normalement l'espace d'entrée est d'une dimension supérieure à 2. Au départ, les valeurs des poids (des connexions entre entrée et colonnes) sont issues d'un tirage aléatoire autour d'une valeur « centrale » - dans cet exemple 0,5. A la présentation de la première donnée d'entrée, l'une des colonnes gagne la compétition. Cette gagnante va voir ses poids modifiés de telle façon que la prochaine fois que la même entrée sera présentée, elle gagnera la compétition avec une plus grande probabilité. Si l'on représente la position de cette colonne dans l'espace[7] des poids (Wx et Wy), on distingue que celle-ci

7Il y a deux poids par colonne, un pour chaque entrée .

s'est extirper de l'ensemble des colonnes (fig. 3A). En comparant avec la position de la donnée représentée dans l'espace des entrées (X,Y) (fig. 3B), nous constatons que la colonne se dirige vers la donnée d'entrée. Nous ne devrions pas comparer une position dans l'espace des poids à une position dans l'espace des entrées – quel lien peut-il y avoir entre une force de connexion (qui traduit un nombre de molécule de neuro-transmetteur) et une entrée (représentée par les niveaux d'activité électrique sur des neurones) ? A priori aucun, sauf dans ce cas précis ! Le raccourci mathématique proposé par T. Kohonen pour accélérer la convergence de l'apprentissage (réduire sa durée) établit un lien direct entre les entrées et les poids. La distance (qui détermine qui la colonne gagnante) est calculée comme la somme des valeurs absolues des différences entre la valeur de chaque entrée et le poids de chaque connexion entre cette entrée et la colonne.

La modification des poids de la colonne gagnante est de rapprocher ses poids des valeurs des entrées. La nouvelle valeur d'un poids est égale à l'ancienne valeur plus une partie de la différence entre la valeur de l'entrée et la valeur du poids. Evidemment, si la partie retenue est en fait l'entièreté de la différence, alors la nouvelle valeur du poids est égale à la valeur de l'entrée. Ceci garanti bien évidemment que lors d'une future présentation en entrée de la même entrée, alors ce sera bien cette colonne qui gagnera (puisque sa distance sera nulle). En pratique, comme le nombre de colonnes est très inférieur au nombre des exemples présentés en entrée, chaque colonne doit représenter un grand nombre d'exemples (et non pas le dernier). La « partie » de la différence à retenir est donc généralement de quelques % (20% est une valeur habituelle).

La figure 3C montre le déplacement de 4 colonnes supplémentaires (en sus de la gagnante). Il s'agit des colonnes voisines de la gagnante – celles au Nord, au Sud, à l'est et à l'Ouest. L'idée ici est que les colonnes voisines sont tellement proches de la gagnante qu'elles subissent l'influence de son excitation (à la gagnante) et donc modifient leurs poids – à un degré moindre. La partie de la différence sera donc plus petite que celle conservée pour la gagnante, par exemple 5%.

Une loi de modification des poids entre neurones dans le cerveau a été proposée en 1942 par Donald Hebb, un neuropsychologiste canadien. Elle est connue sous le nom de « loi de Hebb » et a été depuis validée par l'observation. Cette loi dit que lorsque un neurone en active un autre,

alors la connexion entre les deux est renforcée afin de permettre une meilleure efficacité de l'action excitatrice du premier sur le second. Dans le cadre de la TnC, la loi de Hebb a été complétée comme nous le verrons au chapitre suivant.

La figure 4 montre sur le même graphique à la fois la position des données fournies lors de l'apprentissage et les positions des poids des colonnes. Les segments de droite montrent les liens de voisinage entre les colonnes (Nord, Sud, Est, Ouest). L'affichage choisi montre la situation après la première données (A), la seconde (B), la cinquième (C) et la centième (D). La carte retenue est constitué de 16 colonnes corticales réparties en 4 lignes (de 4). Comme nous pouvons nous en rendre compte, la distribution des données résulte d'un tirage aléatoire uniforme dans l'espace des entrées.

La figure 5 montre le résultat d'un apprentissage d'une carte de 16 colonnes lorsque la distribution n'est pas uniforme. Plus il y a de données à un endroit et plus il y a de colonnes pour représenter cet endroit. C'est exactement la propriété que montre l'Homoncule de Penfield.

La figure 6 montre le résultat d'un apprentissage lorsque le nombre de dimensions de l'espace des entrées est de 3, avec une carte à deux dimensions. On voit bien la carte se replier de manière à passer dans l'ensemble du volume. Cela nous donne une bonne idée de ce que la carte réalise dans des espaces multidimensionnels, quelque soit le nombre de dimensions. Certaines applications des cartes auto-organisatrices impliquent un nombre de dimensions de plusieurs centaines, ou milliers.

La figure 7 montre le résultat d'un apprentissage d'une carte à une dimension (un « fil ») dans un espace à deux dimensions. Comme précédemment (fig. 6) la carte se replie pour que chaque zone soit représentée avec le moins d'erreurs possibles lors de la projection des données. Deux donnée voisines doivent (le plus possible) être représentées par la même colonne, ou deux colonnes voisines. Les zones peu denses en données sont moins bien représentées – ce qui est normal puisqu'elles généreront au final moins d'erreurs que les zones plus denses.

Comme le montre la figure 8, à l'issue de l'apprentissage chaque zone de l'espace d'entrée est représentée par une colonne. Pour délimiter ces zones, il suffit de positionner les droites orthogonales passant par les points milieux entre deux colonnes. Ces zones sont appelées champs

récepteurs en biologie, et zones ou polygones de Voronoï en Mathématiques. Toute données appartenant à un champ récepteur active la colonne qui le représente. Chaque colonne est donc au centre de chaque champ récepteur.

Dans les exemples que nous venons de voir, les données sont à deux ou trois dimensions. Les domaines de valeurs des données sont identiques, mais cela ne signifie absolument pas que les données X et Y sont de même nature. Par exemple, X pourrait être l'âge des membres d'une population et Y pourrait indiquer le nombre des enfants de chaque individu (de la population). Cela fonctionnerait tout aussi bien si X représentait le niveau de gris et Y la « forme » (triangle, carré,etc.). La carte de Kohonen représenterait alors les éléments (forme et niveau de gris) des objets présentés durant l'apprentissage (triangle noir, carré gris clair, etc.).

Le programme Python implémentant une carte auto-organisatrice, son apprentissage et permettant de générer les figures 3 à 8, est donné ci-dessous. C'est un programme très court au niveau du nombre de lignes de code et en même temps très générique. Ce sont les données (et l'apprentissage) qui en feront un programme utile. Comme pour tous les programmes de cet ouvrage, les commentaires sont indiqués dans le code, soit sur la même ligne, soit par blocs s'il s'agit d'informations plus détaillées.

Le Python utilisé est la version 2.7. L'utilisation de la bibliothèque « plot » est nécessitée par l'affichage graphique – pas par la carte auto-organisatrice. Il est fait appel à un générateur de nombre aléatoire dès lors que l'on ne sait pas choisir a priori les bonnes valeurs – comme dans le cas des valeurs initiales des poids. Il vaut mieux une initialisation aléatoire – vraisemblablement ni bonne, ni mauvaise – qu'une initialisation choisie et malencontreusement très mauvaise.

Notez que cette initialisation aléatoire signifie que deux expérimentations successives, même pour des données d'apprentissage identiques, ne généreront pas deux fois le même résultat. Lorsque le résultat n'est pas conforme à nos attentes, alors il peut être utile de se rendre au chapitre 12 (maladies mentales) pour découvrir ce qu'un tel (mauvais) résultat peut suggérer comme pathologie mentale.

```
# -*- coding:Utf-8 -*-
from random import random
import matplotlib.pylab as plt
from pylab import *
'''
Programme d'une carte auto-organisatrice avec apprentis-
sage de données 2D et affichage des données et de la
carte. C. Touzet. 08/12/2013
La 1ere ligne permet d'utiliser des accents dans les
commentaires
Les 3 suivantes concernent des functions additionnelles
nécessaires
'''

print " carte auto-organisatrice"
Wx=[]  # poids des colonnes connectes a l'entree X
Wy=[]# idem Y
X_vect=[];Y_vect=[]
jj=kk=5 # nombre de plots (graphiques pour l'affichage)
cote=7 # racine carre du nombre de N (la carte de Kohnonen
sera donc carré)
nbN=cote*cote # nb de colonnes
alpha=0.2 #alpha (20% pour le gagnant)
beta=0.05 # beta (5% pour les voisins)
aff=10*cote*cote  #  on  affiche  tous  les  aff  points
(fonction du nbre de colonnes)
nbit = (jj*kk)*aff # nb iterations apprentissage

###fonction pour l'affichage des données et de la carte##

def affiche (ii) :
      #divise la carte par ligne
      Wxx=[]; Wyy=[]
      for c in range (cote) :
          Wxx.append([])
          Wyy.append([])
      for c in range(cote) :
          for i in range (c*cote,  (c+1)*cote):
              Wxx[c].append(Wx[i])
              Wyy[c].append(Wy[i])
      #divise la carte par colonne
      Wxxx=[]; Wyyy=[]
      for c in range (cote) :
          Wxxx.append([])
```

```
            Wyyy.append([])
        for c in range(cote) :
            for i in range (c,  (c+cote*cote),  cote):
                Wxxx[c].append(Wx[i])
                Wyyy[c].append(Wy[i])

        plt.subplot(jj,kk,ii) # les différents graphiques
             axvspan(xmin=0.01,  xmax=0.99,  ymin=0,  ymax=1,
alpha=0) #les axes des graphiques
        #plt.plot(X_vect,  Y_vect, '.') #à commenter si on ne
désire pas afficher les exemples apprentissage
        for c in range(cote) :
            plt.plot(Wxx[c],Wyy[c],'o-')
            plt.plot(Wxxx[c],Wyyy[c])

#####main : la carte de Kohonen########

for i in range(nbN): # initialisation aleatoire entre 0.45
et 0.55 des poids
        Wx.append(random()*0.10+0.45)
        Wy.append(random()*0.10+0.45)

plt.figure() # création de la fenêtre pour les graphiques
ii=0 # variable utilisée pour afficher la carte à inter-
valle fixé
for  i  in  range  (nbit):  #boucle  du  nombre  d'itérations
souhaitées
        #print "Itération n° ", i
        a = alpha - alpha * i / nbit  # decroissance lineaire
de la valeur du pas d apprentissage pour le gagnant
        b = beta - beta * i / nbit # idem pour les voisins
         if (i%(aff) == 0)   :   ii=ii+1; affiche (ii) # pour
l'affichage

        #les données d'apprentissage X et Y
         X=random() #l'entrée X est un nombre aléatoire entre
0 et 1, tirage uniforme
        Y=random() #idem pour Y
           X_vect.append(X)  #  X  est  stocké  pour  affichage
eventuel
        Y_vect.append(Y) # idem pour Y

          # Calcul des distances (pour la recherche de la
colonne gagnante)
        D=[] # distance des colonnes
```

```
      for j in range (nbN):
          d=abs(X - Wx[j]) + abs(Y - Wy[j])
          D.append(d)

        # quel est l' « index » de la colonne ayant la
distance minimale ?
      mini=D[0]
      index=0
      for j in range (nbN):
          if (D[j]<mini):
              mini=D[j]
              index=j

      #print "Apprentissage : modification des poids"
      for k in range (nbN):
          if (k==index): #du gagnant"
              Wx[k] = Wx[k] + a * (X - Wx[k])
              Wy[k] = Wy[k] + a * (Y - Wy[k])
          if(index%cote!=0):
              if (k==index-1) : #du voisin Ouest
                  Wx[k] = Wx[k] + b * (X - Wx[k])
                  Wy[k] = Wy[k] + b * (Y - Wy[k])
          if ((index+1)%cote!=0):
              if (k==index+1) : #du voisin Est
                  Wx[k] = Wx[k] + b * (X - Wx[k])
                  Wy[k] = Wy[k] + b * (Y - Wy[k])
          if ((index+cote)<nbN):
              if (k==index+cote): #du voisin Nord
                  Wx[k] = Wx[k] + b * (X - Wx[k])
                  Wy[k] = Wy[k] + b * (Y - Wy[k])
          if ((index-cote)>=0):
              if (k==index-cote): #du voisin Sud
                  Wx[k] = Wx[k] + b * (X - Wx[k])
                  Wy[k] = Wy[k] + b * (Y - Wy[k])
plt.show() #affichage
```

Proprioception : homoncule sensoriel et moteur
représentation 2D, etc...
Auto-organisation
Modèle de carte corticale
colonne corticale / neurone
algorithme
exemples de fonctionnement :
2D, 1D, 3D, homoncule

Rappel des avantages :
Projection 2D respectant fréquence et distribution
Apprentissage permanent (optimisation des ressources)
Capacité de prédiction (voisinage)
Bases d'apprentissage
Fusion de données (Coiton et Gilhodes)
co-activation, co-occurrence, neurones mirroirs
Bases d'apprentissage

Exemple de carte apprenant à répartir les pays (données CIA), programme permettant des entrées multiples et un affichage de labels.

CHAPITRE 4 : COMPACTER LES REPRESENTATIONS

Programmes moteurs : séquences d'actions (marcher, évitement d'obstacles, suivi de murs)

synergie entre cartes corticales sensorielle et sensori-motrice, apprentissage permanent

Associer les cartes :
synthèse instantanée de comportements pour robot mobile

Rappel des avantages :
Projection 2D respectant fréquence et distribution
Apprentissage permanent (optimisation des ressources)
Capacité de prédiction (voisinage)
Possibiliter de (re)connaître une séquence (trajectoire)

Bases d'apprentissage

Génération de comportements complexes comme une succession de comportements élémentaires, avec passage automatique d'un comportement au suivant dès l'extinction du (sous-)but. Il y a donc une hiérarchie de cartes sensorielles et sensori-motrices.

L'hippocampe comme l'endroit où les index spatiaux et temporels sont stockés, lesquels permettent la navigation entre les (sous-)buts. La verbalisation peut faire la même chose (fournir un accès cohérent sur les différentes cartes de (sous)-buts.

La mémoire autobiographique utilise ces index, qui disparaissent

naturellement avec un usage répété (incohérence des index pour le même souvenir) et deviennent donc la mémoire culturelle.

Bases d'apprentissage

Labelliser, comprendre, parler

Labelliser des situations : (cf. chapitre Coiton appliqué à la parole)

Selon le niveau dans la hiérarchie, la labellisation permet d'identifier et nommer des concepts de plus en plus abstraits.

Bases d'apprentissage (supervision parentale + Hebb)

Comprendre : s'approprier des labellisations (en les rejouant pour soi : connexions récurrentes)

CHAPITRE 5 : TAILLE ET NOMBRE DES CARTES CORTICALES

Le cas de la lecture, un exemple de fonctions cognitives (haut niveau d'abstraction, plusieurs étages corticaux – hiérarchie de cartes corticales) avec ses 7 étages de traitements

étage 1 : on-off

étage 2 : barre de constaste

étage 3 : angle

étage 4 : lettres (supervisé)

étage 5 : LETTRE (supervisé)

étage 6 : bigramme

étage 7 : mot (carte corticale)

Apprentissage des lettres avec multimodalité (sensorielle et sensori-motrice ; oeil et main)

Quelques améliorations nécessaires pour traiter de grands volumes de données :

carte des 50 000 mots (orthographe)

Certains étages de neurones réalisent des traitements standards (histo-grammes, filtres)

L'avantage des méthodes syllabiques par rapport aux méthodes globales ou semi-globales (supervision au niveau lettres, syllabes (bigrammes)

Les connexions récurrentes sont utiles pour améliorer la performance (prédiction / vérification) et réalisent les capacités attentionnelles (endo et exo).

Le degré de certitude est fonction de la distance entre prédiction et vérification

Influence de l'ordre des mots chronologique dans la mémorisation (position/voisinage), ce qui est conforme à l'idée véhiculée par Hebb+ (à savoir que l'ordre, comme la durée inter-exemple, sont fondamentals)

CHAPITRE 6 : NORMALISER L'EFFICACITE SYNAPTIQUE

Intelligence et sommeil

Durant l'éveil, renforcement de connexions excitatrices à longue distance (Hebb), avec de temps à autre un « éclair de génie » qui consiste à connecter ensemble deux activités neuronales jusqu'à lors séparées

Sommeil lent et sommeil paradoxal (normalisation des forces de connexions excitatrices)

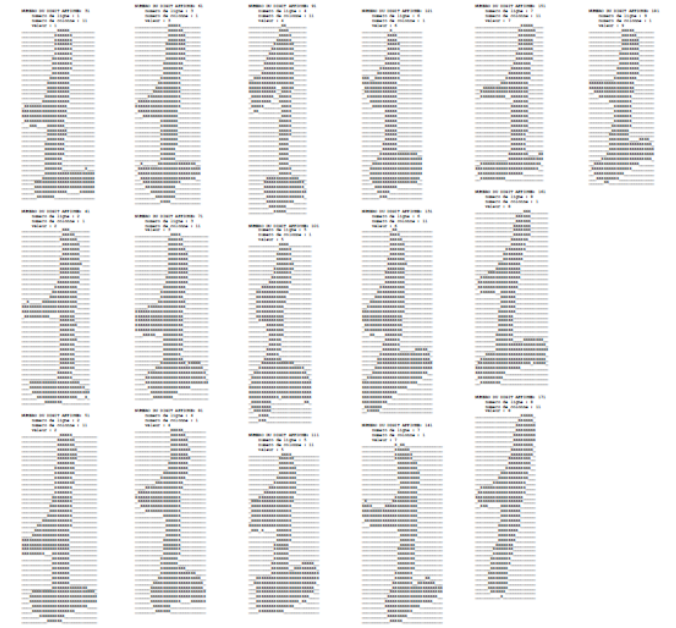
algorithme

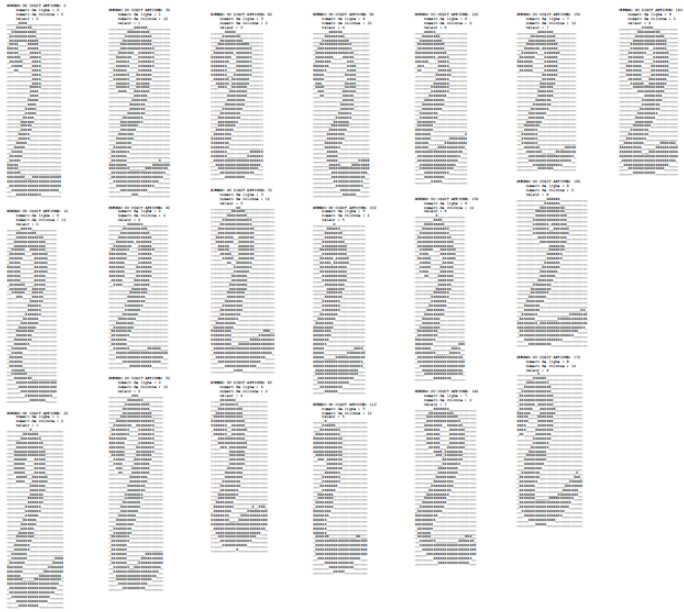
exemple de fonctionnement :

Réduction du nombre de connexions (i.e., de l'arbre de recherche)

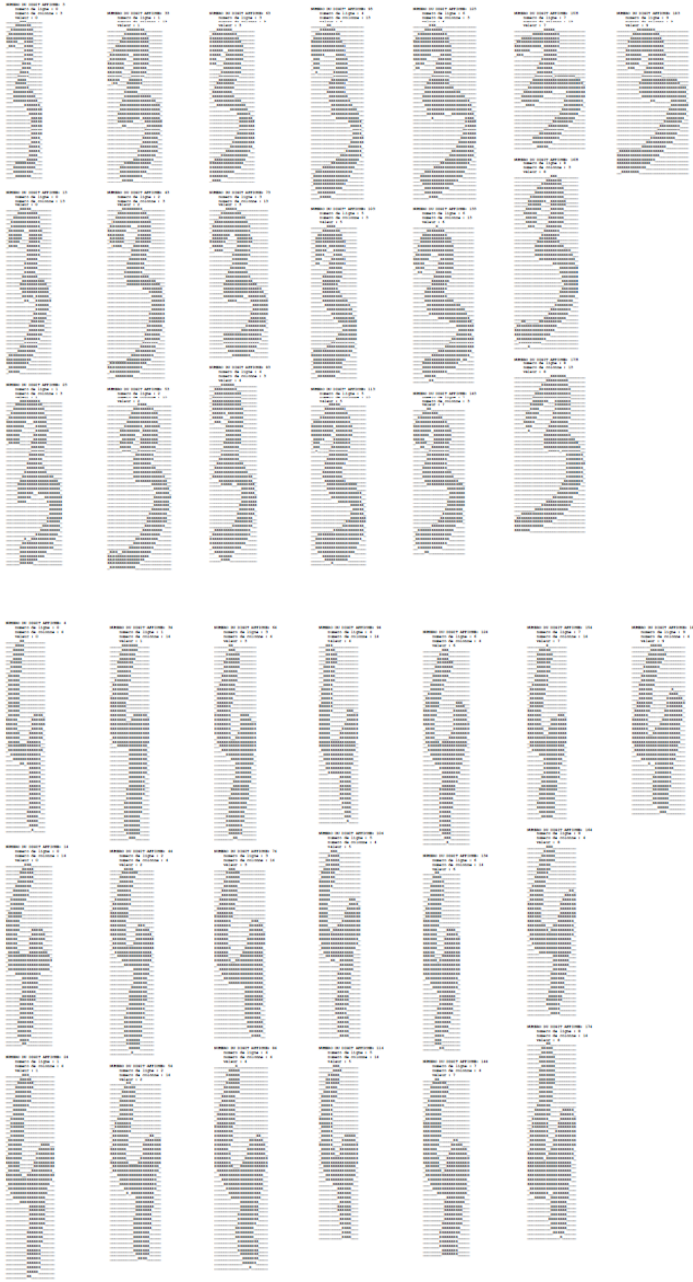
ANNEXE 1 : Les 190 chiffres manuscrits

I. HISTORIQUE

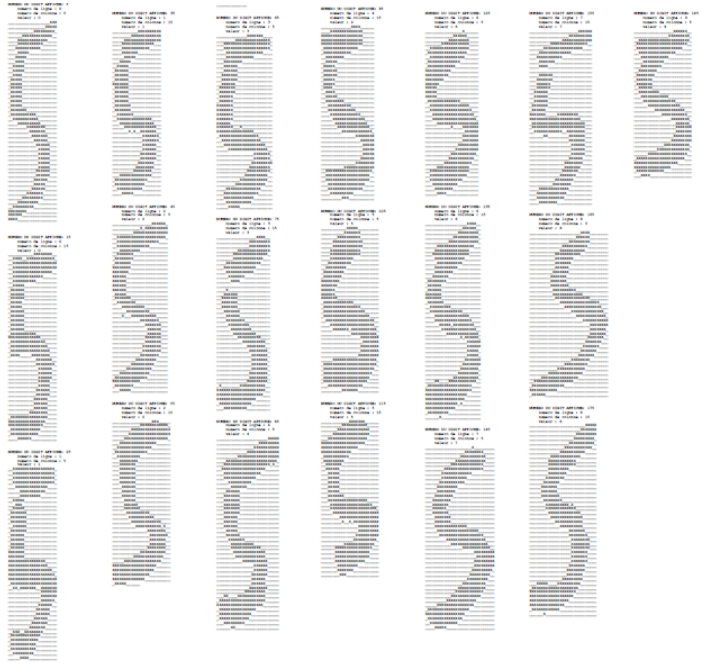




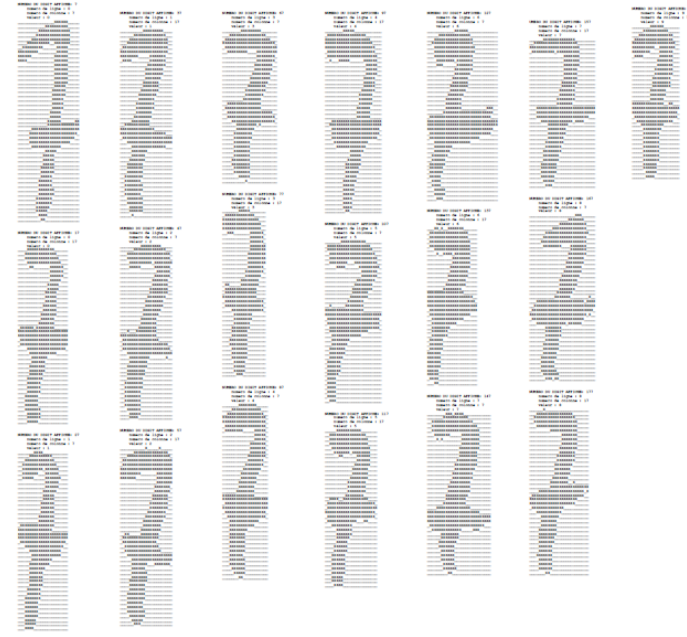
LES RESEAUX DE NEURONES ARTIFICIELS !

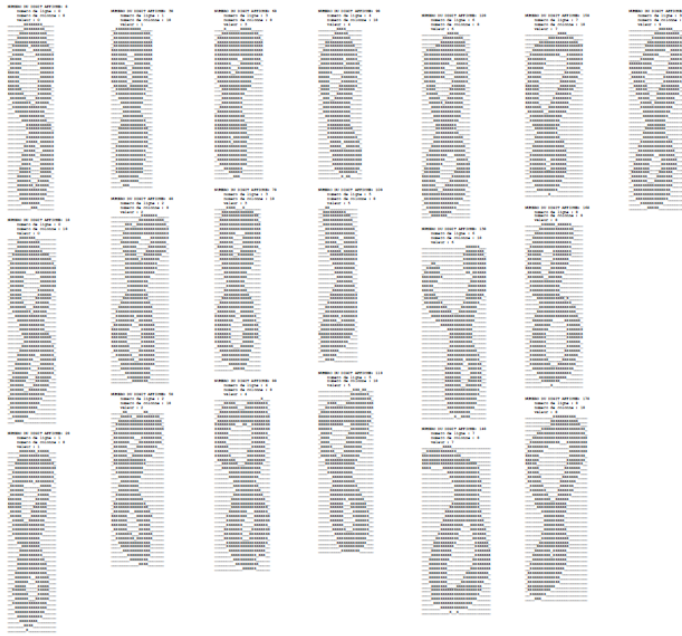


I. HISTORIQUE



LES RESEAUX DE NEURONES ARTIFICIELS !





I. HISTORIQUE

ANNEXE 2 : BA.py

```
BA = [ # Base d'Apprentissage :
  # 19 x 10 chiffres manuscrits sur une matrice de 3 colonnes x 5 lignes
  [12,42,39,35,7,39,29,0,41,38,0,39,18,41,26],
  [0,28,0,12,55,0,29,41,0,0,35,0,10,62,42],
  [42,34,0,19,34,0,3,35,0,30,11,0,54,36,31],
  [21,27,11,0,25,8,8,27,20,0,0,39,6,21,27],
  [16,19,0,36,9,0,41,20,44,5,22,36,0,11,30],
  [18,34,23,43,0,0,30,34,7,0,14,35,26,34,9],
  [1,38,17,30,13,0,48,0,0,50,27,35,20,40,60],
  [31,49,39,4,20,33,11,54,44,7,55,6,4,53,0],
  [18,46,35,47,1,57,17,46,37,1,43,55,11,49,32],
  [31,45,46,46,9,63,42,44,58,0,0,49,6,43,30],
  [37,38,3,37,11,47,43,0,53,40,0,45,22,45,29],
  [0,44,0,5,58,0,36,52,0,0,45,0,30,60,49],
  [33,41,19,46,22,26,6,43,7,36,18,0,26,38,33],
  [28,45,16,9,34,11,0,39,41,0,19,33,28,37,2],
  [9,39,0,40,11,3,45,42,44,2,50,5,4,46,0],
  [34,43,30,52,0,0,41,39,18,0,14,40,35,48,15],
  [13,36,2,34,13,0,44,15,3,44,19,42,24,36,37],
  [17,28,36,0,18,26,35,50,45,16,39,3,31,23,0],
  [35,53,23,51,13,52,17,55,26,33,40,52,52,42,10],
  [33,55,28,53,23,64,33,36,50,0,17,55,48,38,6],
  [15,59,29,35,10,51,41,0,49,48,2,50,26,65,31],
  [0,48,0,1,52,0,68,49,0,0,45,1,42,75,55],
  [26,51,3,23,41,9,8,52,0,44,13,16,55,52,48],
  [48,49,11,17,42,9,15,56,47,3,2,54,11,51,26],
  [25,33,0,47,7,11,60,42,60,5,43,25,0,47,10],
  [34,49,28,49,5,0,61,35,23,10,20,50,34,45,6],
  [0,40,17,19,39,0,47,35,13,52,22,59,51,50,28],
  [27,38,23,0,28,19,40,61,44,16,38,1,35,16,0],
  [31,54,43,46,17,30,32,61,9,30,41,27,27,46,17],
  [40,53,19,57,23,47,31,39,54,0,2,59,52,59,18],
  [1,23,38,38,29,56,43,0,50,49,1,50,19,44,36],
  [0,42,6,0,56,2,38,62,0,0,54,0,16,67,47],
  [25,64,9,48,46,7,12,50,0,50,9,1,50,49,35],
  [25,53,11,40,40,6,18,58,36,6,7,47,12,45,21],
  [14,46,1,42,25,11,25,45,50,0,30,38,0,42,19],
  [13,38,26,46,1,0,38,18,7,2,19,42,24,41,15],
  [0,25,24,11,41,3,34,17,1,47,32,52,29,36,39],
  [32,51,42,0,45,23,28,66,13,30,41,13,35,22,0],
  [37,39,35,41,26,32,18,48,17,22,42,26,26,44,12],
```

```
  [45,39,3,54,46,42,23,31,60,0,2,58,34,52,27],
  [26,43,16,53,17,40,43,0,47,52,0,57,34,41,34],
  [0,17,27,0,27,34,28,48,28,0,21,22,20,54,33],
  [21,47,7,51,32,15,10,50,2,32,24,0,32,36,30],
  [46,45,18,11,51,9,1,25,45,0,0,55,27,45,24],
  [19,31,0,51,17,21,18,37,47,0,33,33,0,45,14],
  [23,38,33,53,2,0,31,28,7,0,18,39,34,42,6],
  [0,18,26,7,41,5,25,30,0,34,37,29,17,37,36],
  [15,40,49,0,22,40,20,56,15,32,43,16,40,19,0],
  [14,39,51,18,51,39,23,63,5,56,33,22,29,54,15],
  [16,64,20,37,56,55,7,29,59,0,3,60,39,53,26],
  [11,62,38,43,13,53,52,0,53,56,1,55,47,59,20],
  [0,48,10,0,60,2,37,75,0,19,58,0,35,71,42],
  [26,60,22,54,38,30,9,62,4,50,22,0,56,51,46],
  [11,59,27,14,51,22,0,20,56,0,11,57,41,55,10],
  [18,26,0,45,12,25,42,39,56,1,32,38,0,33,28],
  [15,44,41,43,24,0,23,23,0,1,24,49,40,49,23],
  [7,41,0,24,33,0,38,32,7,39,37,54,24,50,46],
  [38,43,52,0,9,50,24,58,25,20,44,23,29,31,0],
  [31,45,42,41,42,19,30,41,0,45,36,12,19,54,16],
  [14,62,19,57,40,70,26,40,56,0,0,70,46,59,34],
  [13,54,10,43,17,45,54,0,52,53,4,56,19,49,22],
  [0,32,17,1,56,12,30,59,3,1,54,7,31,64,26],
  [29,45,11,43,38,17,10,50,2,40,18,0,33,40,38],
  [38,41,7,14,45,5,21,33,19,0,10,48,33,41,15],
  [15,36,0,46,19,7,36,40,52,0,36,33,3,50,7],
  [28,48,28,49,3,0,56,17,2,14,30,50,21,51,22],
  [0,33,18,14,37,1,41,25,0,48,34,28,21,38,49],
  [34,45,38,0,21,36,25,51,21,20,39,8,21,29,0],
  [38,46,40,45,38,27,34,39,0,49,46,1,28,41,0],
  [0,41,37,0,56,63,0,5,55,0,0,51,38,46,45],
  [21,72,29,59,11,57,49,0,56,53,0,64,34,59,31],
  [0,40,7,0,52,10,49,70,2,5,56,6,24,64,39],
  [23,51,2,35,54,6,9,52,0,53,20,10,50,56,51],
  [36,51,3,23,42,7,8,61,20,2,11,53,7,56,40],
  [20,22,0,49,9,15,50,18,58,13,32,56,0,4,49],
  [7,40,46,21,22,1,18,42,6,0,8,54,32,46,33],
  [3,44,10,27,31,0,47,41,25,52,13,55,34,46,45],
  [26,29,28,0,6,53,28,40,34,14,45,8,12,35,0],
  [24,41,37,31,39,40,13,49,8,44,32,40,27,41,22],
  [9,39,47,28,29,51,7,37,48,0,4,53,44,52,24],
  [10,71,32,46,11,57,55,0,48,47,1,61,22,59,45],
  [0,29,19,23,61,13,20,58,0,0,58,8,17,64,52],
  [42,37,0,25,50,0,12,42,0,42,19,22,41,53,38],
  [43,49,2,12,58,3,30,52,22,0,9,55,41,48,18],
  [20,29,0,56,14,8,59,44,70,3,51,39,0,43,30],
  [18,43,58,48,32,1,44,21,1,3,42,32,51,55,11],
  [0,19,22,13,41,5,32,28,3,45,39,47,30,41,34],
  [28,45,38,0,14,43,17,50,24,30,48,23,21,33,0],
  [44,35,55,48,12,49,10,63,31,21,46,55,13,56,38],
  [18,53,43,41,13,63,18,49,57,0,6,60,41,57,22],
  [10,55,13,40,17,40,43,0,38,28,0,46,13,53,28],
  [0,38,17,36,58,22,13,23,16,0,24,10,3,60,37],
  [34,46,7,40,13,23,2,40,10,17,37,5,2,48,52],
  [42,44,0,10,42,6,0,44,34,0,6,45,37,59,15],
  [20,17,0,32,9,3,40,17,44,15,40,29,0,10,25],
```

```
[41,52,14,40,0,0,24,49,17,0,6,50,26,55,34],
[0,24,30,13,39,1,37,21,2,42,33,39,15,37,45],
[49,55,37,0,2,46,21,39,51,12,59,13,0,40,0],
[15,39,20,36,24,38,26,36,18,12,39,28,1,33,41],
[20,49,10,50,39,25,34,35,42,0,0,48,33,58,44],
[0,35,32,19,36,43,46,1,47,35,10,44,46,61,10],
[0,35,8,28,57,2,11,40,0,31,54,32,51,26,29],
[18,49,0,25,60,2,27,36,0,49,9,20,42,58,28],
[20,42,8,11,37,8,5,50,18,0,11,45,33,53,19],
[30,16,0,41,4,31,38,46,38,1,34,19,0,39,7],
[15,51,8,50,17,0,47,38,16,7,18,63,43,63,40],
[0,19,34,14,39,3,41,40,22,45,18,54,27,45,32],
[37,47,38,0,24,37,43,55,38,45,15,0,34,0,0],
[18,36,23,7,33,40,0,29,33,15,39,40,30,39,15],
[25,55,1,42,40,33,36,69,53,2,34,82,81,71,22],
[0,51,16,30,44,44,54,0,45,45,7,58,51,63,14],
[0,35,3,33,60,3,29,48,0,6,50,34,42,59,35],
[41,25,0,31,33,0,39,13,0,45,0,2,38,61,41],
[32,49,6,23,56,5,12,56,20,0,13,52,33,57,26],
[14,28,0,34,9,0,32,31,45,7,34,30,0,15,24],
[25,52,23,44,12,0,33,26,18,3,23,62,10,55,26],
[2,43,18,26,32,0,45,6,0,50,51,37,29,47,56],
[26,49,26,0,23,28,35,57,40,18,28,0,31,13,0],
[24,34,46,46,36,39,29,52,3,23,49,32,8,49,40],
[35,49,20,57,30,61,27,48,66,0,9,67,68,76,25],
[7,44,13,32,20,29,37,0,35,30,4,46,16,48,24],
[0,30,0,31,45,0,15,38,0,3,43,14,29,51,37],
[39,36,0,11,42,2,28,40,0,46,17,30,28,53,22],
[57,45,0,22,58,0,25,51,54,0,7,61,31,64,24],
[9,44,0,44,9,32,31,44,50,0,50,13,0,39,0],
[16,40,21,49,22,0,31,48,30,0,13,56,26,53,17],
[0,21,45,8,48,7,34,23,0,46,36,24,13,42,31],
[47,74,2,4,57,0,48,65,51,59,30,11,38,0,0],
[24,47,37,44,25,46,15,56,20,43,34,50,50,49,29],
[19,65,25,50,11,61,31,59,50,10,13,60,63,68,18],
[9,69,20,53,40,43,58,0,61,52,10,63,35,65,18],
[0,36,0,9,63,0,58,59,0,19,63,51,64,36,10],
[24,39,7,37,28,27,3,46,9,42,27,3,38,43,48],
[34,46,0,27,55,0,23,53,14,0,16,64,26,50,23],
[15,27,0,37,17,15,40,26,48,7,32,33,0,36,9],
[16,44,43,49,21,0,30,50,31,0,7,58,57,52,17],
[0,25,13,14,46,3,35,43,18,55,17,55,34,47,35],
[32,47,29,0,30,29,45,56,39,38,13,0,40,0,0],
[24,0,67,49,9,57,9,58,35,0,56,47,0,53,44],
[26,65,11,59,21,57,35,56,64,0,1,69,31,70,37],
[27,67,25,56,15,50,52,0,50,50,1,56,36,47,26],
[0,44,4,16,65,10,26,58,5,0,57,9,2,75,40],
[27,48,14,12,44,23,15,59,2,54,13,0,37,44,50],
[37,39,5,16,51,6,12,40,32,0,10,51,30,45,10],
[26,27,0,46,15,6,43,35,53,5,40,40,0,48,24],
[8,53,27,47,17,0,28,53,38,0,13,62,47,62,12],
[10,53,7,40,18,1,58,35,20,58,38,58,40,39,16],
[33,48,21,0,39,31,4,58,5,53,61,36,21,25,0],
[45,53,15,0,54,43,17,62,50,52,19,63,38,62,30],
[0,60,4,0,92,41,0,49,55,0,0,68,58,82,64],
[3,56,41,37,10,46,39,0,43,40,0,41,57,32,39],
```

```
  [0,6,31,0,30,29,20,36,18,0,28,18,31,44,32],
  [20,48,4,27,39,5,3,40,0,32,17,0,22,40,32],
  [28,44,0,10,40,0,23,49,53,11,6,45,4,43,7],
  [7,37,1,30,21,0,38,31,44,3,38,29,2,46,9],
  [19,39,36,39,2,0,58,40,25,2,9,44,22,48,6],
  [0,42,23,17,39,0,40,22,6,57,26,65,45,43,26],
  [28,51,17,0,34,11,23,62,32,13,40,0,24,18,0],
  [24,41,45,49,8,48,43,43,12,37,57,8,36,50,7],
  [13,54,48,47,13,56,26,38,55,0,1,51,31,45,25],
  [0,18,47,10,52,65,49,6,56,57,1,53,48,53,16],
  [0,24,19,5,55,2,29,56,0,3,46,17,54,55,36],
  [10,62,8,7,57,12,4,48,0,40,14,10,56,57,44],
  [8,51,28,20,38,24,0,43,62,0,2,57,44,48,15],
  [11,26,0,44,4,27,49,33,48,6,34,30,0,39,8],
  [0,44,16,9,55,1,0,33,61,0,5,52,45,51,6],
  [0,12,42,1,46,4,26,54,30,53,17,58,28,41,47],
  [9,47,39,3,26,39,21,62,34,34,52,16,27,24,0],
  [31,51,30,50,45,13,25,54,25,42,14,53,29,40,29],
  [6,60,16,45,40,62,40,51,57,2,16,58,48,52,11],
  [17,93,43,58,23,66,76,0,69,65,10,61,67,70,9],
  [0,3,50,1,35,46,31,56,22,4,55,3,18,36,0],
  [37,35,0,38,49,0,25,34,0,49,7,25,48,58,26],
  [5,55,30,25,37,37,0,26,54,0,4,59,44,53,17],
  [13,40,1,26,32,18,54,39,60,58,59,41,0,32,24],
  [0,22,55,21,55,2,8,62,27,0,14,53,58,60,10],
  [0,29,15,12,48,0,42,34,18,61,41,60,51,51,16],
  [23,47,21,0,36,24,27,65,33,33,37,0,30,16,0],
  [27,55,57,52,10,54,31,59,6,30,78,3,64,42,0],
  [51,40,7,60,49,38,19,36,61,0,1,68,46,59,30],
  [9,59,32,50,25,70,56,0,57,53,0,67,34,47,31],
  [0,26,14,16,47,16,10,48,1,0,56,21,30,46,20],
  [13,54,21,30,38,33,2,55,12,22,38,7,43,53,43],
  [31,66,1,15,61,13,0,34,62,0,23,45,57,45,1],
  [6,41,0,32,19,27,54,35,51,20,54,28,0,43,9],
  [26,33,36,41,19,0,23,37,19,0,3,53,36,43,24],
  [0,42,31,13,47,0,40,38,1,52,43,26,48,59,17],
  [33,42,24,4,26,26,21,44,20,22,42,10,14,35,0],
  [7,52,54,50,22,52,18,73,9,35,56,22,47,64,10],
  [19,74,20,59,37,79,35,60,68,0,12,72,69,88,14]]

BAD = [ # BA : les valeurs désirés pour chacun des 10 perceptrons
    [1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 1, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 1, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 1, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 1, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 1, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 1]]

BA2 = [
  #[0, 20, 40, 60, 90, 110, 140, 150, 170, 180], # après manipulation
  [0, 10, 20, 30, 40, 50, 60, 70, 80, 90],
  [1, 11, 21, 31, 41, 51, 61, 71, 81, 91],
```

```
  [2, 12, 22, 32, 42, 52, 62, 72, 82, 92],
  [3, 13, 23, 33, 43, 53, 63, 73, 83, 93],
  [4, 14, 24, 34, 44, 54, 64, 74, 84, 94],
  [5, 15, 25, 35, 45, 55, 65, 75, 85, 95],
  [6, 16, 26, 36, 46, 56, 66, 76, 86, 96],
  [7, 17, 27, 37, 47, 57, 67, 77, 87, 97],
  [8, 18, 28, 38, 48, 58, 68, 78, 88, 98],
  [9, 19, 29, 39, 49, 59, 69, 79, 89, 99]]
  # BA2 contient les numeros des exemples de BT1 (taille 190) utilises
pour la base d'apprentissage
  # le premier vecteur contient les 10 exemples du 0, le suivant les 10
exemples du 1, etc .
```