Les réseaux neuronaux :

1. First Attempts**:** There were some initial simulations using formal logic. McCulloch and Pitts (1943) developed models of neural networks based on their understanding of neurology. These models made several assumptions about how neurons worked. Their networks were based on simple neurons which were **considered to be binary devices with fixed thresholds**. The results of their model were simple logic functions such as "a or b" and "a and b". Another attempt was by using computer simulations. Two groups (Farley and Clark, 1954; Rochester, Holland, Haibit and Duda, 1956). The first group (IBM researchers) maintained closed contact with neuroscientists at McGill University. So whenever their models did not work, they consulted the neuroscientists. This interaction established a **multidisciplinary** trend which continues to the present day.
2. Promising & Emerging Technology**:** Not only was neuroscience influential in the development of neural networks, but psychologists and engineers also contributed to the progress of neural network simulations. Rosenblatt (1958) stirred considerable interest and activity in the field when he designed and developed the **Perceptron**. The Perceptron had three layers with the middle layer known as the association layer. This system could learn to connect or associate a given input to a random output unit.  
   Another system was the **ADALINE** (ADAptive LInear Element) which was developed in 1960 by Widrow and Hoff (of Stanford University). The ADALINE was an analogue electronic device made from simple components. The method used for learning was different to that of the Perceptron; it employed the **Least-Mean-Squares (LMS) learning rule.**
3. Period of Frustration & Disrepute**:** In 1969 Minsky and Papert wrote a book in which they generalized the limitations of single layer Perceptron’s to multilayered systems. In the book they said: *"...our intuitive judgment that the extension (to multilayer systems) is sterile*". The significant result of their book was to eliminate funding for research with neural network simulations. The conclusions supported the disenchantment of researchers in the field. As a result, considerable prejudice against this field was activated.
4. Innovation**:** Although public interest and available funding were minimal, several researchers continued working to develop **neuromorphically based computational methods** for problems such as pattern recognition.   
   During this period several paradigms were generated which modern work continues to enhance. Grossberg's (Steve Grossberg and Gail Carpenter in 1988) influence founded a school of thought which **explores resonating algorithms**. They developed the ART (Adaptive Resonance Theory) networks based on biologically plausible models. Anderson and Kohonen developed associative techniques independent of each other. Klopf (A. Henry Klopf) in 1972 developed a basis for learning in artificial neurons based on a **biological principle for neuronal learning called heterostasis.**   
   Werbos (Paul Werbos 1974) developed and used the **back-propagation learning metho**d, however several years passed before this approach was popularized**. Back-propagation nets** are probably **the most well-known and widely applied** of the neural networks today. In essence**, the back-propagation net is a Perceptron** with multiple layers, a different threshold function in the artificial neuron, and a more robust and capable learning rule.  
   Amari (A. Shun-Ichi 1967) was involved with theoretical developments: he published a paper which established a **mathematical theory for a learning basis (error-correction method)** dealing with adaptive patern classification. While Fukushima (F. Kunihiko) developed a step wise trained multilayered neural network for interpretation of handwritten characters. The original network was published in 1975 and was called **the Cognitron.**
5. Re-Emergence**:** Progress during the late 1970s and early 1980s was important to the re-emergence on interest in the neural network field. Several factors influenced this movement. For example, comprehensive books and conferences provided a forum for people in diverse fields with specialized technical languages, and the response to conferences and publications was quite positive. The news media picked up on the increased activity and tutorials helped disseminate the technology. Academic programs appeared and courses were introduced at most major Universities (in US and Europe). Attention is now focused on funding levels throughout Europe, Japan and the US and as this funding becomes available, several new commercial with applications in industry and financial institutions are emerging.
6. Today**:** Significant progress has been made in the field of neural networks-enough to attract a great deal of attention and fund further research. Advancement beyond current commercial applications appears to be possible, and research is advancing the field on many fronts. Neurally based chips are emerging and applications to complex problems developing. Clearly, today is a period of transition for neural network technology.

Sources <http://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.html#Appendix%20A%20-%20Historical%20background%20in%20detail>

Les réseaux de neurones, en tant que système capable d'apprendre, mettent en œuvre le principe de l'induction, c’est-à-dire l'apprentissage par l'expérience. Par confrontation avec des situations ponctuelles, ils infèrent un système de décision intégré dont le caractère générique est fonction du nombre de cas d'apprentissages rencontrés et de leur complexité par rapport à la complexité du problème à résoudre. Par opposition, les systèmes symboliques capables d'apprentissage, s'ils implémentent également l'induction, le font sur base de la logique algorithmique, par complexification d'un ensemble de règles déductives.

Grâce à leur capacité de **classification** et de **généralisation**, les réseaux de neurones sont généralement utilisés dans des problèmes de nature statistique, tels que la classification automatique de codes postaux ou la prise de décision concernant un achat boursier en fonction de l'évolution des cours. Autre exemple, une banque peut créer un jeu de données sur les clients qui ont effectué un emprunt constitué : de leur revenu, de leur âge, du nombre d’enfants à charge… et s’il s’agit d’un bon client. Si ce jeu de données est suffisamment grand, il peut être utilisé pour l’entraînement d’un réseau de neurones. La banque pourra alors présenter les caractéristiques d’un potentiel nouveau client, et le réseau répondra s’il sera bon client ou non, en généralisant à partir des cas qu’il connaît.

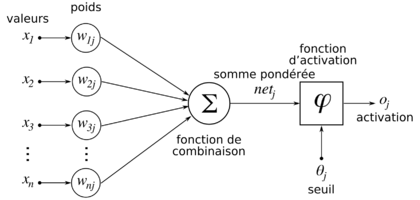
Si le réseau de neurones fonctionne avec des nombres réels, la réponse traduit une probabilité de certitude. Par exemple : 1 pour « sûr qu’il sera un bon client », -1 pour « sûr qu’il sera mauvais client », 0 pour « aucune idée », 0,9 pour « presque sûr qu’il sera bon client ».

Le réseau de neurones ne fournit pas toujours de règle exploitable par un humain. Le réseau reste souvent une boîte noire qui fournit une réponse quand on lui présente une donnée, mais le réseau ne fournit pas de justification facile à interpréter.

Les réseaux de neurones sont réellement utilisés, par exemple :

* pour la classification d’espèces animales par espèce étant donnée une analyse ADN.
* reconnaissance de motif ; par exemple pour la Reconnaissance optique de caractères (OCR), et notamment par les banques pour vérifier le montant des chèques, par La Poste pour trier le courrier en fonction du code postal, etc. ; ou bien encore pour le déplacement automatisé de robots mobiles autonomes.
* approximation d’une fonction inconnue.
* modélisation accélérée d’une fonction connue mais très complexe à calculer avec exactitude ; par exemple certaines fonctions d’inversions utilisées pour décoder les signaux de télédétection émis par les satellites et les transformer en données sur la surface de la mer.
* estimations boursières :
  + apprentissage de la *valeur d’une entreprise* en fonction des indices disponibles : bénéfices, endettements à long et court terme, chiffre d’affaires, carnet de commandes, indications techniques de conjoncture. Ce type d’application ne pose pas en général de problème
  + tentatives de prédiction sur la périodicité des cours boursiers. Ce type de prédiction est très contesté pour deux raisons, l’une étant qu'il n'est pas évident que le cours d’une action ait de façon tout à fait convaincante un caractère périodique (le marché anticipe en effet largement les hausses comme les baisses prévisibles, ce qui applique à toute périodicité éventuelle une variation de période tendant à la rendre difficilement fiable), et l’autre que l’avenir prévisible d’une entreprise détermine au moins aussi fortement le cours de son action, si ce n'est plus encore que peut le faire son passé ; les cas de Pan Am, Manufrance ou IBM permettent de s’en convaincre.
* modélisation de l'apprentissage et amélioration des techniques de l'enseignement.
* en météorologie, pour la classification de conditions atmosphériques et la prévision statistique du temps.
* en auscultation des ouvrages hydrauliques, pour la compréhension physique des phénomènes de déplacements, sous-pressions et débits de fuite.

### Structure du réseau

[](http://commons.wikimedia.org/wiki/File:ArtificialNeuronModel_francais.png)

Structure d'un neurone artificiel. Le neurone calcule la somme de ses entrées puis cette valeur passe à travers la fonction d'activation pour produire sa sortie.

Un réseau de neurones est en général composé d'une succession de couches dont chacune prend ses entrées sur les sorties de la précédente. Chaque couche (i) est composée de Ni neurones, prenant leurs entrées sur les Ni-1 neurones de la couche précédente. À chaque synapse est associé un poids synaptique, de sorte que les Ni-1 sont multipliés par ce poids, puis additionnés par les neurones de niveau i, ce qui est équivalent à multiplier le vecteur d'entrée par une matrice de transformation. Mettre l'une derrière l'autre les différentes couches d'un réseau de neurones reviendrait à mettre en cascade plusieurs matrices de transformation et pourrait se ramener à une seule matrice, produit des autres, s'il n'y avait à chaque couche, la fonction de sortie qui introduit une non linéarité à chaque étape. Ceci montre l'importance du choix judicieux d'une bonne fonction de sortie : un réseau de neurones dont les sorties seraient linéaires n'aurait aucun intérêt.

Au-delà de cette structure simple, le réseau de neurones peut également contenir des boucles qui en changent radicalement les possibilités mais aussi la complexité. De la même façon que des boucles peuvent transformer une logique combinatoire en logique séquentielle, les boucles dans un réseau de neurones transforment un simple dispositif de reconnaissance d'entrées en une machine complexe capable de toutes sortes de comportements.

### Fonction de combinaison

Considérons un neurone quelconque.

Il reçoit des neurones en amont un certain nombre de valeurs via ses connexions synaptiques, et il produit une certaine valeur en utilisant une **fonction de combinaison**. Cette fonction peut donc être formalisée comme étant une fonction vecteur-à-scalaire, notamment :

* Les réseaux de type **MLP** (Multi-Layer Perceptron) calculent une combinaison linéaire des entrées, c’est-à-dire que la fonction de combinaison renvoie le produit scalaire entre le vecteur des entrées et le vecteur des poids synaptiques.
* Les réseaux de type **RBF** (Radial Basis Function) calculent la distance entre les entrées, c’est-à-dire que la fonction de combinaison renvoie la norme euclidienne du vecteur issu de la différence vectorielle entre les vecteurs d’entrées.

### Fonction d’activation

La **fonction d’activation** (ou **fonction de seuillage**, ou encore **fonction de transfert**) sert à introduire une non-linéarité dans le fonctionnement du neurone.

Les fonctions de seuillage présentent généralement trois intervalles :

1. en dessous du seuil, le neurone est non-actif (souvent dans ce cas, sa sortie vaut 0 ou -1) ;
2. aux alentours du seuil, une phase de transition ;
3. au-dessus du seuil, le neurone est actif (souvent dans ce cas, sa sortie vaut 1).

Des exemples classiques de fonctions d’activation sont :

1. La fonction sigmoïde.
2. La fonction tangente hyperbolique.
3. La fonction de Heaviside.

La logique bayésienne, dont le théorème de Cox-Jaynes formalise les questions d’apprentissage, fait intervenir aussi une fonction en S qui revient de façon récurrente : ev(p) = 10 \log \left(\frac{p}{1-p}\right)

### Propagation de l’information

Ce calcul effectué, le neurone propage son nouvel état interne sur son axone. Dans un modèle simple, la fonction neuronale est simplement une fonction de seuillage : elle vaut 1 si la somme pondérée dépasse un certain seuil ; 0 sinon. Dans un modèle plus riche, le neurone fonctionne avec des nombres réels (souvent compris dans l’intervalle [0,1] ou [-1,1]). On dit que le réseau de neurones passe d'un état à un autre lorsque tous ses neurones recalculent en parallèle leur état interne, en fonction de leurs entrées.

## Type d’apprentissage

* **L'apprentissage supervisé** : si les *classes* sont prédéterminées et les *exemples* connus, le système apprend à classer selon un *modèle* de classement ; on parle alors d'apprentissage supervisé (ou d'analyse discriminante). Un expert (ou *oracle*) doit préalablement étiqueter des exemples. Le processus se passe en deux phases. Lors de la première phase (hors ligne, dite d'*apprentissage*), il s'agit de déterminer un modèle des données étiquetées. La seconde phase (en ligne, dite de *test*) consiste à prédire l'étiquette d'une nouvelle donnée, connaissant le modèle préalablement appris. Parfois il est préférable d'associer une donnée non pas à une classe unique, mais une probabilité d'appartenance à chacune des classes prédéterminées (on parle alors d'apprentissage supervisé probabiliste).

Ex : L'analyse discriminante linéaire ou les SVM en sont des exemples typiques. Autre exemple : en fonction de *points communs* détectés avec les symptômes d'autres patients connus (les *exemples*), le système peut catégoriser de nouveaux patients au vu de leurs analyses médicales en risque estimé (probabilité) de développer telle ou telle maladie.

* **L'apprentissage non supervisé** (ou classification automatique). Quand le système ou l'opérateur ne disposent que d'exemples, mais non d'étiquettes, et que le nombre de classes et leur nature n'ont pas été prédéterminés, on parle d'apprentissage non supervisé ou *clustering*. Aucun expert n'est requis. L'algorithme doit découvrir par lui-même la structure plus ou moins *cachée* des données. Le partitionnement de données, *data clustering* en anglais, est un algorithme d'apprentissage non supervisé.  
  Le système doit ici — dans l'espace de description (la somme des données) — cibler les données selon leurs attributs disponibles, pour les classer en groupe *homogènes* d'exemples. La similarité est généralement calculée selon une fonction de distance entre paires d'exemples. C'est ensuite à l'opérateur d'associer ou déduire du sens pour chaque groupe et pour les motifs (*patterns* en anglais) d'*apparition* de groupes, ou de groupes de groupes, dans leur « espace ». Divers outils mathématiques et logiciels peuvent l'aider. On parle aussi d'analyse des données en régression (ajustement d'un modèle par une procédure de type moindres carrés ou autre optimisation d'une fonction de *coût*). Si l'approche est probabiliste (c'est-à-dire que chaque exemple, au lieu d'être classé dans une seule classe, est caractérisé par un jeu de probabilités d'appartenance à chacune des classes), on parle alors de « *soft clustering* » (par opposition au « *hard clustering* »).  
  Cette méthode est souvent source de sérendipité.

Ex : Pour un épidémiologiste qui voudrait dans un ensemble assez large de victimes de cancer du foie tenter de faire émerger des hypothèses explicatives, l'ordinateur pourrait différencier différents groupes, que l'épidémiologiste chercherait ensuite à associer à divers facteurs explicatifs, origines géographique, génétique, habitudes ou pratiques de consommation, expositions à divers agents potentiellement ou effectivement toxiques (métaux lourds, toxines telle que l'aflatoxine, etc.).

* **L'apprentissage semi-supervisé**. Effectué de manière probabiliste ou non, il vise à faire apparaître la distribution sous-jacente des *exemples* dans leur espace de description. Il est mis en œuvre quand des données (ou « étiquettes ») manquent… Le modèle doit utiliser des exemples *non étiquetés* pouvant néanmoins renseigner.

Ex : En médecine, il peut constituer une aide au diagnostic ou au choix des moyens les moins onéreux de tests de diagnostic.

* **L'apprentissage partiellement supervisé** (probabiliste ou non), quand l'étiquetage des données est partiel. C'est le cas quand un modèle énonce qu'une donnée n'appartient pas à une classe *A*, mais peut-être à une classe *B* ou *C* (*A, B* et *C* étant 3 maladies par exemple évoquées dans le cadre d'un diagnostic différentiel).
* **L'apprentissage par renforcement** : l'algorithme apprend un comportement étant donné une observation. L'action de l'algorithme sur l'environnement produit une valeur de retour qui guide l'algorithme d'apprentissage.

Ex : L'algorithme de Q-learning est un exemple classique.

## Théorème approximation universelle ou théorème de Cybenko

n the mathematical theory of artificial neural networks, the **universal approximation theorem** states that a feed-forward network with a single hidden layer containing a finite number of neurons (i.e., a multilayer perceptron), can approximate continuous functions on compact subsets of **R**n, under mild assumptions on the activation function. The theorem thus states that simple neural networks can *represent* a wide variety of interesting functions when given appropriate parameters; it does not touch upon the algorithmic learnability of those parameters.

One of the first versions of the theorem was proved by George Cybenko in 1989 for sigmoid activation functions.

Kurt Hornik showed in 1991 that it is not the specific choice of the activation function, but rather the multilayer feedforward architecture itself which gives neural networks the potential of being universal approximators. The output units are always assumed to be linear. For notational convenience, only the single output case will be shown. The general case can easily be deduced from the single output case.

The theorem in mathematical terms:

Let φ(·) be a nonconstant, bounded, and monotonically-increasing continuous function. Let *Im* denote the *m*-dimensional unit hypercube [0,1]*m*. The space of continuous functions on *Im* is denoted by *C*(*Im*). Then, given any function *f* ∈ *C*(*Im*) and є > 0, there exist an integer *N* and real constants *αi*, *bi* ∈ **R**, *wi* ∈ **R***m*, where *i* = 1, ..., *N* such that we may define:


  F( x ) =
  \sum_{i=1}^{N} \alpha_i \varphi \left( w_i^T x + b_i\right)


as an approximate realization of the function *f* where *f* is independent of φ; that is,


  | F( x ) - f ( x ) | < \varepsilon


for all *x* ∈ *Im*. In other words, functions of the form *F(x)* are dense in *C*(*Im*).

## Learning Algorithms of Neural Network: Least Mean-Square (LMS) Algorithm

|  |
| --- |
| Learning Algorithms of Neural Network: Least Mean-Square(LMS) Algorithm  By : Khalid Isa (PhD Student)  The LMS algorithm was introduced by Widrow and Hoff in 1959. It has several names, including the Widrow-Hoff rule and also Delta rule. LMS is an example of supervised learning algorithm in NN similar with the perceptron learning algorithm (refer to the previous article, May 2011). In the perceptron learning algorithm, the algorithm trains the perceptron until it correctly classifies the output of the training set but LMS uses another termination criterion in order to train the perceptron. So instead of training the perceptron until a solution is found, another criterion is to continue training while the Mean-Square Error (MSE) is greater than a certain value. This is the basis for the LMS algorithm.  LMS is a fast algorithm that minimizes the MSE. The MSE is the average of the weighted sum of the error for N training sample which defined as:  http://nurn.eng.usm.my/imagesnurn/bulletin/vol4_june/p4_1.jpg  Where R is the output of the perceptron and Cj is the current test inputs.  In order to train the perceptron by using LMS, we can iterate the test set, taking a set of inputs, computing the output and then using the error to adjust the weight. This process can be done either randomly by the test set, or for each test of the set in succession. The learning rule of LMS is given as:  http://nurn.eng.usm.my/imagesnurn/bulletin/vol4_june/p4_2.jpg  The learning rule adjusts the weight based on the error (R-C or expected output minus actual output). Once the error is calculated, the weights are adjusted by a small amount, p in the direction of the input, E. This has the effect of adjusting the weights to reduce the output error.  The implementation of LMS is very simple. Initially, the weights vector is initialized with small random weights. The main repetition then randomly selects a test, calculates the output of the neuron, and then calculates the error. Using the error, the formula of learning rule is applied to each weight in the vector. Then continues the repetition to check the MSE to see if it has reached an acceptable value, and if so, exit and emit the computed truth table for the neuron. The following C code shows the LMS learning algorithm.  **double weights[NUM\_WEIGHTS]; #define MAX\_TESTS 4  const training\_data\_t training\_set[MAX\_TESTS]={{-1.0,-1.0,-1.0},{-1.0,1.0,-1.0},{1.0,-1.0,-1.0},{1.0,1.0,1.0}};  double compute\_output(int test)  { double result;  result = ((training\_set[test].a\*weights[0])+(training\_set[test].b\*weights[1])+(1.0 \* weights[2])); return(results); }  int classify(int test)  { double result; result = co mpute\_output(test);  if (result > 0.0) return 1; else return-1; }  double MSE(void)  { int test; double sum = 0.0;  for (test = 0; test <MAX\_TESTS; test++)  { sum += sqr(training\_set[test].expected-compute\_output)test)); } return (sum/(double)MAX\_TESTS); }  int main()  { int i, test; double result, error; RANDINIT();  for (i=0; <iNUM\_WEIGHTS;i++)  { weights[i]=RAND\_WEIGHT; }  while (MSE() > 0.26  { test = RANDMAX(MAX\_TESTS); result = compute\_output(test); error = training\_set[test].expected - result;  weights[0] += (p\*error\*training\_set[test].a); weights[1]+=(p\*error\*training\_set[test].b);  weights[2] += (p\*error); printf("mse = %g\n",MSE(i)); }  for(i=0;i <MAX\_TESTS;i++)  { printf("%g AND %g = %d\n", training\_set[i].a, training\_set[i].b, classify(i)); } return 0; }**  **References**  1. N.P. Padhy, Artificial Intelligence and Intelligent Systems, Oxford University Press, 2005.  2. M. Tim Jones, Artificial Intelligence A Systems Approach, Infinity Science Press, 2008.  3. Ham, F. and Kostanic, I., Principles of Neurocomputing for Science and Engineering, McGraw Hill, New York, NY, 2001. |