Les Perceptrons

Ikram Chraibi Kaadoud
Alexandra Delmas-Mitchell
Brice Miard
Maxime Vellard

Master 2 Sciences Cognitives et Ergonomie
UFR Sciences et Modélisation
Université Bordeaux Segalen

Le 10 décembre 2013

Sommaire

Introduction

a) Caractéristiques

Un neurone est une cellule constituant le système nerveux spécialisé dans la communication et le traitement d'informations. Cette cellule est excitable, c'est-à-dire qu'un stimulus peut entraîner la formation d'un signal bioélectrique (influx nerveux). Celui-ci est transmis à d'autres neurones afin de les activer au travers de ses différentes connexions appelées synapses.

Chaque neurone est composé :

- D'un corps cellulaire comportant le noyau ;
- De nombreuses ramifications de type dendritiques (d'où proviennent les informations) ;
- D'un axone, qui correspond à sa sortie. C'est par cette voie que les informations sont diffusées. Sa longueur peut atteindre 1 mètre pour seulement 1 à 15 micromètres de diamètre. Il est entouré par des cellules de Schwann (séparées par les nœuds de Ranvier) qui confèrent une gaine de myéline protectrice tout le long de l'axone. De plus, cette gaine permet d'améliorer la conductance du signal :

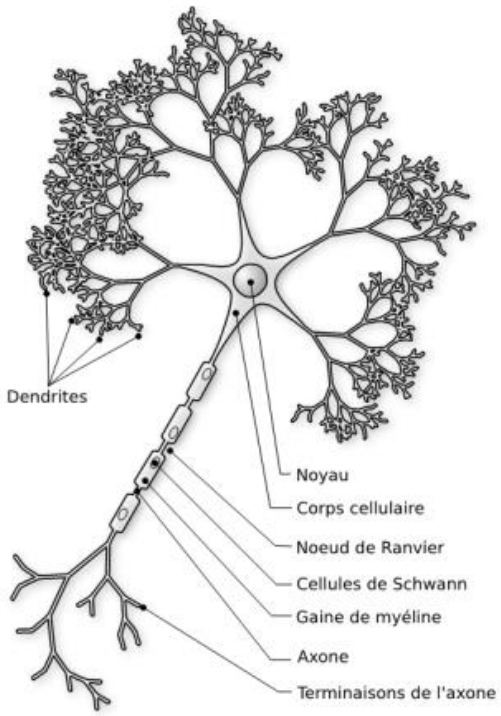


Figure 1 : Structure d'un neurone

Il existe plusieurs types de neurones (pyramide, panier, Purkinje, etc.) avec des fonctionnements différents (sensoriel, moteur, inter-neurones, etc.). Etant interconnectés, ils forment des réseaux. On estime à près de 100 milliards le nombre de neurones dans le cerveau humain. Ils sont donc capables de créer un réseau très complexe, avec un nombre moyen de 10 000 connexions par neurones.

Le cerveau humain, cet organe qui coordonne tout le système nerveux central et périphérique, possède de grandes capacités d'apprentissage. Dans le but de comprendre le fonctionnement du cerveau, les chercheurs utilisent souvent des techniques d'imagerie numérique.

Ces techniques ont l'inconvénient d'être couteuses et souvent peu précises. Afin de mieux comprendre le fonctionnement interne des réseaux, ils font aussi appels à des réseaux de neurones artificiels.

Ces réseaux servent donc d'une part à comprendre et à valider les modèles des réseaux naturels. Et d'autres parts, ils servent à faire des calculs complexes sans essayer d'imiter le fonctionnement humain. La théorie des réseaux de neurones a commencé à émerger au milieu du XXème siècle, notamment avec l'introduction du perceptron en 1957 par Franck Rosenblatt.

Un perceptron est un réseau de neurones artificiels qui permet d'apprendre des fonctions « automatiquement », composé d'une entrée (x_i) et d'une sortie (y). Ces entrées possèdent des poids (w_i) et dont dépend la sortie.

L'objectif du perceptron est d'imiter la stimulation d'un neurone (sortie) par des neurones voisins (entrées). Tout d'abord, considérons des neurones binaires : un neurone a la valeur 1 s'il est actif, 0 sinon. L'état du neurone de sortie correspond à la réponse du perceptron tandis que les p neurones d'entrée déterminent la variable fonctionnelle sur laquelle opère le perceptron. Ce dernier apparaît alors comme une fonction que l'on peut schématiser sous cette forme.

$$f : \begin{cases} \{0,1\}^p \longrightarrow \{0,1\} \\ (x_1, \ldots, x_p) \longrightarrow y = f(x_1, \ldots, x_p) \end{cases}$$

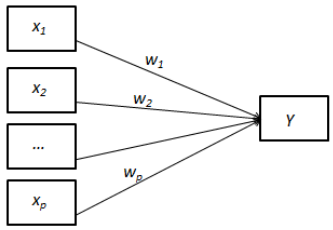


Figure 2 : Schéma d'un perceptron simple

Pour déterminer comment la fonction f va se comporter, on considère que les neurones d'entrée peuvent stimuler le neurone de sortie par le biais de poids synaptiques w_1, …, w_n. On somme alors les poids synaptiques des neurones actifs pour calculer le stimulus généré par une entrée (x_1, …, x_n) : $\sum_{j=1}^{p} w_j * x_j$

Si cette quantité est supérieure à un seuil θ, alors on considère que le neurone de sortie est activé, dans ce cas la fonction f, dite de transfert, est précisée :

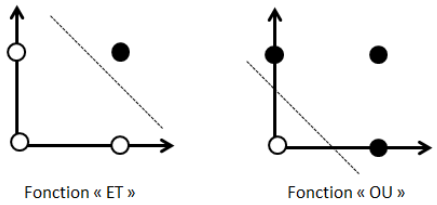
$$f(x1, \ldots, xp) = \begin{cases} 1 \text{ si } \sum_{j=1}^{p} w_j * x_j \geq \theta \\ 0 \text{ sinon} \end{cases}$$

Comme l'action du perceptron est déterminée par les des poids synaptiques et par le seuil d'activation, il faut calculer les valeurs de ces paramètres en fonction de l'action attendue. L'objectif est alors, grâce à un corpus d'apprentissage, de déterminer les valeurs des paramètres pour que le perceptron apprenne correctement. Si la base d'apprentissage n'est pas complète, on espère que, confronté à une nouvelle stimulation le perceptron réagira au mieux par rapport aux exemples initialement appris.

Dans l'espace R^p, l'équation $\sum_{j=1}^{p} w_j * x_j = \theta$ détermine un hyperplan qui sépare le plan initial en 2 semi-espaces. Suivant si les neurones d'entrées (représentés par le p-uplet (x_1, …, x_p)) appartiennent à tel ou tel semi-espace, la valeur prise par f(x_1, …, x_p) sera égale à 0 ou 1.

Il existe 2 sortes de perceptrons : simples et multicouches. Les perceptrons simples sont composés de 2 couches. Ils permettent d'apprendre un échantillon d'exemples si et seulement si le

problème est linéairement séparable, c'est-à-dire que les éléments envoyés sur 0 peuvent être séparés de ceux envoyés sur 1 par un hyperplan, comme c'est le cas pour la fonction « OU » ou « ET » :



Fonction « ET » Fonction « OU »

Pour déterminer son algorithme d'apprentissage, nous disposons d'une base d'exemple que nous souhaitons lui faire apprendre. Pour cela, il faut déterminer les valeurs des poids synaptiques et du seuil. Cela consiste à calculer les poids w_i des neurones d'entrée. On commence par initialiser aléatoirement la valeur de ceux-ci, puis pour chaque exemple $(x_1, \ldots, x_p)$ de la base d'apprentissage. On compare ensuite la réponse fournie à la réponse attendue. Si elles sont égales, on ne modifie pas les poids synaptiques. Cependant, si la réponse fournie ne concorde pas, on ajoute 1 aux poids synaptiques des neurones d'entrée activés. Ceci a pour effet d'accroitre leur influence. A l'opposé, si la réponse fournie correspond, on retire 1 aux poids synaptiques des neurones d'entrée activés. L'algorithme d'apprentissage se poursuit jusqu'à ce que tous les éléments de la base d'apprentissage soient étudiés sans qu'aucun poids synaptique ne soit modifié.

Une fois l'algorithme défini, on cherche à faire « apprendre » notre perceptron. Pour cela, on effectue dans le cas du perceptron simple une descente de gradient qui dépend de l'erreur commise par le perceptron en sortie (c'est-à-dire que la réponse obtenue est différente de celle désirée) calculée au préalable.

Pour définir cette erreur, supposons un ensemble de n exemples. On considère la réponse y_k du réseau et la réponse correcte s_k associée à l'exemple k. L'erreur liée à l'exemple k est donc donnée par: $E_k = (y_k - s_k)$ Cette erreur peut être positive $(y_k > s_k)$ ou négative $(y_k < s_k)$.

Une fois cette erreur déterminée, on effectue la descente de gradient La descente de gradient a pour objectif de minimiser une fonction réelle en construisant par un processus itératif une suite convergeant vers un minimum de celle-ci. En d'autre terme, cela signifie que l'on cherche à réduire l'erreur calculée initialement dans la direction de l'erreur en descendant le long du gradient. Pour cela, on part d'un point initial « bien choisi » et on construit les points suivants en prenant la direction de la plus grande pente durant une distance « bien choisie » appelée « pas » de la descente de gradient. Si l'on considère les entrées x_i du réseau associées respectivement aux poids w_i, alors:

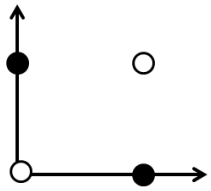
$w_i \leftarrow w_i + \alpha (y_k - s_k) x_i$ où α est le taux d'apprentissage.

Les inconvénients cependant sont multiples :

- le choix du point initial ainsi que celui du pas, sont empiriques
- l'algorithme peut faire converger la suite vers un minimum local
- les termes de la suite peuvent tendre vers l'infini
- la convergence est relativement lente.

Néanmoins, pour l'optimisation des réseaux de neurones, cet algorithme fonctionne plutôt bien.

Le fait que les perceptrons simples ne peuvent discriminer que les ensembles linéairement séparables a pratiquement réduit leur intérêt pratique. En effet, ils ne peuvent pas résoudre des fonctions plus complexes que le « ET » ou le « OU » comme par exemple le « OU exclusif », qui n'est pas linéairement séparable comme on peut le voir dans le schéma ci-dessous :



Fonction « OU exclusif »

Cependant, vers la fin du XXème siècle, apparaît l'idée d'accumuler des couches de neurones intermédiaires avant de produire la réponse finale. Il s'agit du 2ème type de perceptron, le perceptron multicouche qui est composé par définition d'au moins 3 couches dont une cachée.

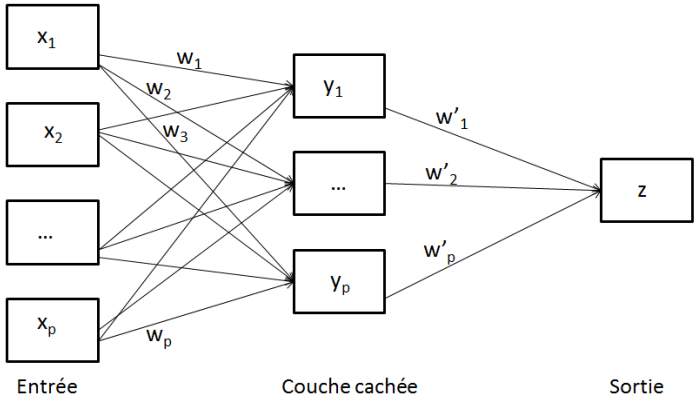


Figure 3 : schéma d'un perceptron multicouche

Par conséquent, on s'est aperçu que plus on augmente le nombre de couche dans le perceptron, et plus on augmente la complexité de séparation entre les 2 semi-espaces :

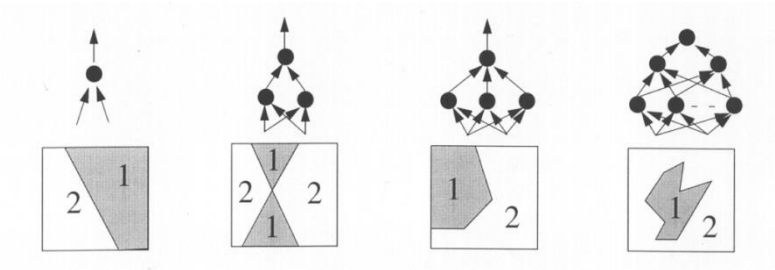


Figure 4 : schéma d'évolution du nombre de couche et des interprétations graphiques

Le problème de l'apprentissage dans les perceptrons multicouches est de connaitre la contribution de chaque poids dans l'erreur globale du réseau. L'algorithme de descente du gradient est adapté à la minimisation de l'erreur produite par un tel réseau. Notamment, il s'interprète comme une rétropropagation de l'erreur en sortie en erreurs partielles commises par les neurones activés dans la couche précédente. La condition est d'avoir une fonction d'activation dérivable car on a besoin de la dérivé pour rétro-propager l'erreur.

Pour concevoir un réseau de neurones, on réfléchit alors préalablement à son architecture en fonction de l'action que l'on veut voir réaliser. On procède ensuite à son apprentissage ce qui est une étape longue compte tenu de la lenteur des algorithmes en jeux. Cependant une fois calibré, la réponse d'un réseau de neurones est quasi immédiate et celui-ci peut être facilement reproduit.

b) Problématique

Aujourd'hui, l'utilisation de réseaux de neurones formels comme le perceptron est devenu monnaie courante. Ils sont devenus indispensables et sont utilisés dans un nombre très important de domaines. Nous verrons quelques exemples de leurs applications plus loin.

Dans le cas de notre projet, on souhaite ici faire de la reconnaissance de formes géométriques, notamment de polygones, pour savoir si un point est à l'intérieur ou à l'extérieur de cette forme. Après avoir spécifié l'architecture du réseau, on déterminera la valeur des poids selon le type de polygone (triangle, carré, hexagone) par un algorithme d'apprentissage et on cherchera à savoir ce qui se passe quand le nombre de côté augmente pour tendre vers un cercle.

I- Historique

a) Evolution du perceptron

C'est au début des années 1940, avec l'émergence du mouvement cybernétique sous l'impulsion de John Von Neumann, Alan Turing ou encore Warren McCulloch, que sont apparus les premiers réseaux de neurones artificiels. En effet, c'est en 1943 que Warren McCulloch et Walter Pitts publient l'article déterminant « *A Logical Calculus Immanent in Nervous Activity* » qui engage un changement conceptuel majeur du mouvement cybernétique. Les auteurs y présentent des neurones formels qui seraient capables de mimer le fonctionnement des neurones biologiques. Le principe est le suivant : un neurone de sortie réalise une somme pondérée de ses entrées, cette somme passe ensuite par une fonction transfert (ici Heaviside, c'est-à-dire « tout ou rien ») qui aboutit à une sortie valant 1 si la fonction dépasse le seuil, 0 sinon. En 1949, le neuropsychologue Donald Hebb dans « *The organisation of behavior : A neuropsychological theory* » émet l'hypothèse que la force d'une connexion synaptique qui unit deux neurones augmentera si ces deux neurones sont actifs en même temps. Cette hypothèse est connue aujourd'hui sous le nom de « loi de Hebb » ou « règle de Hebb » et aura, a postériori, d'importantes répercussions sur les réseaux de neurones artificiels.

En 1956, deux séries de conférences à Cambridge et Dartmouth, où interviennent notamment Noam Chomsky et Marvin Minsky, permettent d'envisager que l'intelligence, en raison de ses similitudes avec le fonctionnement d'un ordinateur, puisse être définie par la *computation* de symboles. Le cognitivisme était né et sa principale application, l'intelligence artificielle, faisait désormais l'objet de nombreuses recherches.

Deux ans plus tard, Frank Rosenblatt présente le perceptron dans « *The perceptron : A probabilistic model for information storage and organization in the brain* ». Ce réseau de neurones formels est composé de deux couches, entrée et sortie, où chaque neurone de la couche d'entrée est

connecté à tous ceux de la couche de sortie. Ce réseau peut apprendre des problèmes linéairement séparables comme des fonctions booléennes simples. En 1960, Bernard Widrow et Marcian Hoff publient « *Adaptive swiching circuits* » où ils exposent le réseau ADALINE (pour ADAptive LInear NEuron). Ici, la fonction d'activation est linéaire et l'algorithme LMS (Least Mean Square) qui minimise le carré de l'erreur permet à ce réseau d'être moins sensible au bruit que le perceptron. Ces concepts seront la base de nombreux travaux dans les années 1960. Cependant, en 1969, Marvin Minsky et Seymour Papert, publient « *Perceptrons* » un ouvrage très critique à l'égard du travail réalisé onze ans plus tôt par Rosenblatt, qui pourtant avait suscité un fort engouement au sein de la communauté scientifique et industrielle. Ils soulignent notamment que le perceptron tel qu'il avait été formulé était incapable d'apprendre une fonction logique comme le « ou exclusif » en raison de la non linéarité du problème. Cela entrainera un désengagement important de la communauté scientifique pour ce domaine.

Au début des années 1980 émerge enfin le connexionnisme après vingt ans d'hégémonie cognitiviste. Ce mouvement s'est développé sous l'effet de deux facteurs : l'avènement de l'informatique accessible à tous et l'adoption du principe d'auto-organisation (intrinsèquement lié à la notion d'émergence) par les physiciens et les mathématiciens, formulé par William Ashby en 1947.

Cette révolution conceptuelle provoque un regain d'intérêt pour les réseaux de neurones formels dans la communauté scientifique. L'année 1982 est marquée par la publication de deux articles majeurs : « *Neural networks and physical systems with emergent computational abilities* » de John Hopfield et « *Self-organized formation of topologically correct feature maps* » de Teuvo Kohonen. Un réseau de Hopfield est une mémoire associative, c'est-à-dire un réseau de neurones artificiels où tous les éléments sont interconnectés et se mettent à jour selon une séquence déterminée. Cela permet l'apprentissage de prototypes sur la base d'exemples et une restitution par la présentation d'un exemple bruité ou partiel. Une carte de Kohonen, quant à elle, est une carte auto-organisatrice, c'est-à-dire un réseau de neurones interconnectés (à l'image du réseau de Hopfield) où est cependant introduite une notion de distance entre ces neurones. Ces réseaux permettent de cartographier un ensemble de données par des regroupements plus ou moins fidèles au réel. La reconquête du perceptron viendra elle des travaux de David Parker en 1985, Yann LeCun en 1986 ainsi que David Rumelhart et David McClelland sur la rétropropagation du gradient de l'erreur. Cette notion permettra de développer le perceptron multicouche, perceptron simple auquel on a rajouté une ou plusieurs couches entre l'entrée et la sortie. Celui-ci utilise le principe de la rétropropagation de l'erreur pour modifier lui-même les poids entre les différentes couches selon ce qui rentre et ce qui sort. La fonction de transfert est devenue, de plus, une fonction sigmoïdale. Grâce à ce dispositif les problèmes du « ou exclusif » et de ceux non linéairement séparables sont résolus.

Depuis, les réseaux de neurones artificiels n'ont cessé de se développer parallèlement aux capacités croissantes de calculs et de stockages des ordinateurs et sont intégrés dans un grand nombre de domaines.

b) Utilisations actuelles

Les différents réseaux de neurones artificiels n'ont pas, bien entendu, tous les mêmes applications. Les réseaux de Hopfield sont utilisés, par exemple, en lecture automatique pour la reconnaissance des caractères, ils permettent notamment à La Poste de trier la destination des différents courriers de manière automatique ou aux banques de lire les chèques sans intervention humaine. Les cartes de Kohonen ont de nombreuses applications, notamment pour les études statistiques de grande envergure où manquent une importante quantité de données. Nous pouvons citer comme exemples l'étude d'Ibbous et Tutin sur la segmentation du parc social en Ile de France ou encore l'étude des données socio-économiques de 96 pays par Patrick Letrémy. Le perceptron multicouche n'est, quant à lui, pas en reste pour les applications concrètes comme pour la compréhension des mécanismes d'interactions entre les ondes électromagnétiques radar avec la surface marine pour les satellites d'observation. Ils servent aussi pour la prédiction bancaire. En effet, à partir de l'expérience de leurs précédents clients, les banques ont calibré un réseau de neurones qui quantifie le risque commis lors d'un accord de prêt. De même, les perceptrons multicouches peuvent s'avérer utiles dans d'autres formes de prédictions comme les consommations d'eau ou d'électricité, la météorologie ou la bourse. Dans l'automatique, on s'en sert pour l'identification et le contrôle de processus comme la commande de robots. Enfin, ils peuvent être utilisés dans la reconnaissance de formes pour le traitement des images. C'est précisément dans cette perspective que s'inscrit notre projet.

II- Méthodologie

Afin de mieux comprendre le fonctionnement du perceptron nous avons voulu faire quelques expérimentations. Pour cela nous avons récupéré le code d'un perceptron multicouche conçu en python par Nicolas Rougier.

Après avoir fait quelques tests sur ce code nous avons conçu d'autres exemples pour mettre en valeurs les paramètres importants de ce programme.

Nous avons voulu par la suite faire apprendre au perceptron diverses formes géométriques simples : le triangle, le carré, l'hexagone et l'octogone. Ces polygones ont été choisis et implémentés dans cet ordre afin de comparer les différences d'apprentissage lorsque l'on crée des formes de plus en plus complexes, c'est-à-dire en augmentant le nombre de côtés.

On pourra ainsi supposer une règle plus générale concernant le lien entre l'apprentissage et le nombre de côtés, notamment lorsque le nombre de côté tend vers l'infini pour créer un cercle.

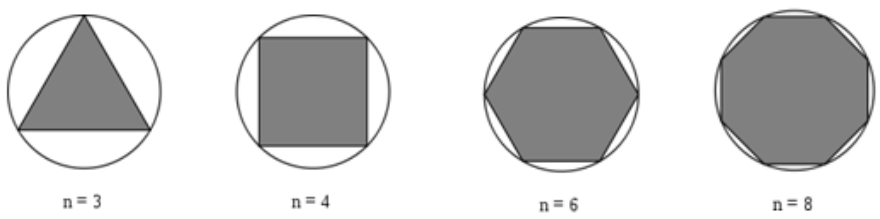


Figure 5 : Schéma des polygones étudiés

Le perceptron utilisé est composé de 3 couches :

- La première est la couche d'entrée, elle est composé de deux neurones, ces neurones prennent comme arguments des coordonnées de points, un tuple (x,y).
- La seconde est la couche cachée qui se compose de n neurones, n variant suivant le nombre de côtés du polygone (voir dessins ci-dessus).
- La dernière couche est la sortie. Elle se compose d'un neurone représentant la probabilité que les coordonnées du point d'entrée se trouvent à l'intérieur ou à l'extérieur du polygone. Nous avons fixé à 1 le fait que le point se trouve dans le polygone et à 0 si il est à l'extérieur.

Dans un premier temps nous avons donc créé des générateurs de points aléatoires à l'intérieur et à l'extérieur de chaque forme.

Afin de pouvoir réaliser l'apprentissage nous avons ajouté à chaque tuple un booléen pour signifier au perceptron si ces coordonnées se trouvent à l'intérieur ou à l'extérieur du polygone.

Exemples de points créés :

((0.9226525952941671, 0.36903036896362185), 1) → Point à l'intérieur

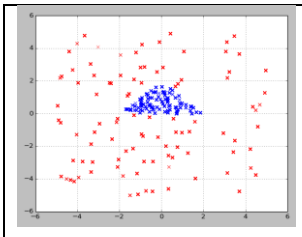
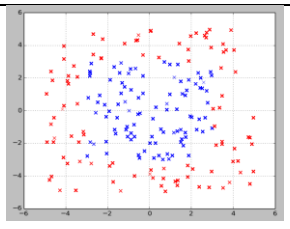
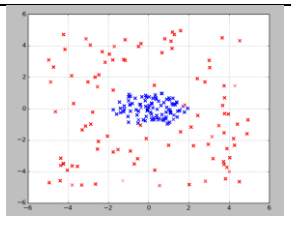
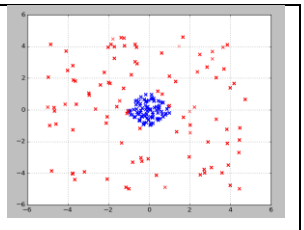
((-1.2178603518408115, 3.655277372248861), 0) → Point à l'extérieur

Pour la phase d'apprentissage, nous plaçons donc un certain nombre de coordonnées (x,y) en entrée et fixons la sortie au booléen correspondant : ce sont les samples.

Lors de l'apprentissage, le perceptron relance un certain nombre de fois chaque sample, à chaque itération, il modifie ces poids en fonction de la sortie attendue et de la sortie réelle : ce sont les epoch (ou cycle).

Pour réaliser la phase de test, nous avons créé un objet à partir de la class samples afin d'obtenir de nouvelles coordonnées aléatoires à l'intérieur et à l'extérieur du polygone.

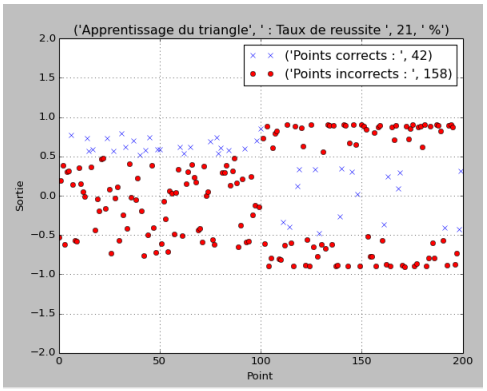
Dans cette phase, les seuls paramètres introduits dans le perceptron sont les coordonnées de nouveaux points aléatoires.

			
Triangle	**Carré**	**Hexagone**	**Octogone**

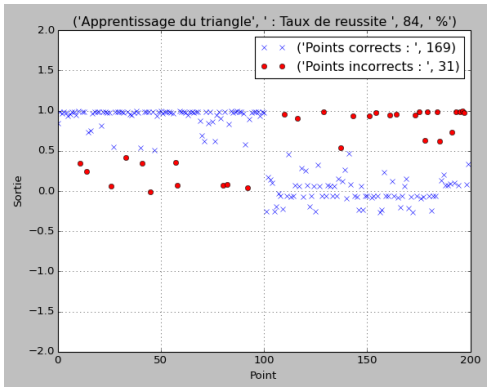
III- Résultats

Les premières simulations que nous avons faites ont porté sur la variation pour une même forme (ici le triangle) du nombre de samples par rapport au nombre de cycle :

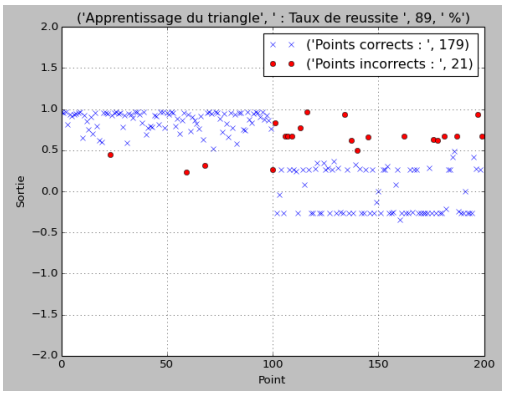
Cycle=2 et nombre de samples = 100



Cycle=2500 et nombre de samples = 100



Cycle=2500 et nombre de samples = 2000



Pour cette forme on remarque que le nombre de samples n'influence pas significativement le taux de réussite, en revanche le nombre de cycles est très important pour l'apprentissage. En effet comme nous l'avions vu, le perceptron modifie ses pondérations à chaque itération. Avec peu de cycles, il n'a pas le temps de maximiser son apprentissage.

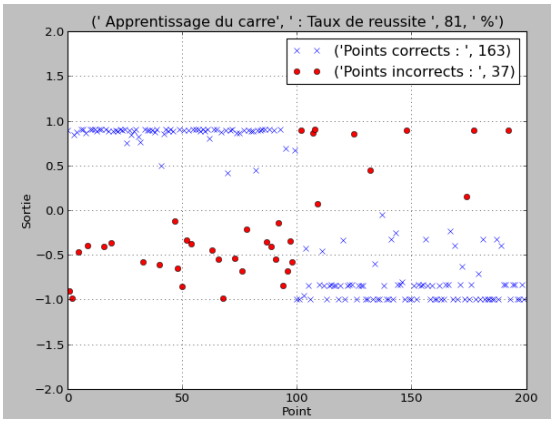
On va maintenant s'intéresser aux différences d'apprentissage entre les formes. Le tableau ci-dessous indique les taux de réussite pour chaque forme suivant le nombre de cycle et de samples :

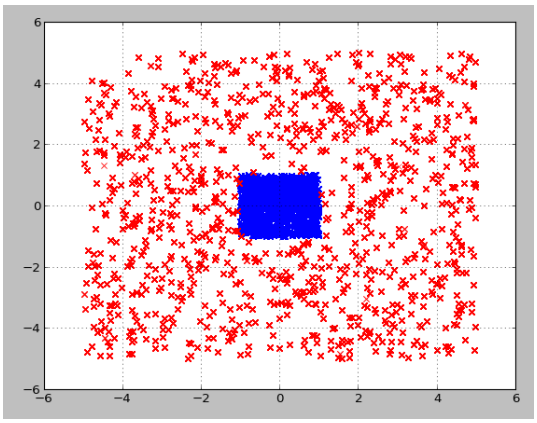
Cycle/Samples	C=2500/s=100	C=2/ s=100	C=2500 / s =2000
Triangle	84%	21 %	89%
Carré	51%	36%	54%
Hexagone	83%	61%	66%
Octogone	83%	56%	82%

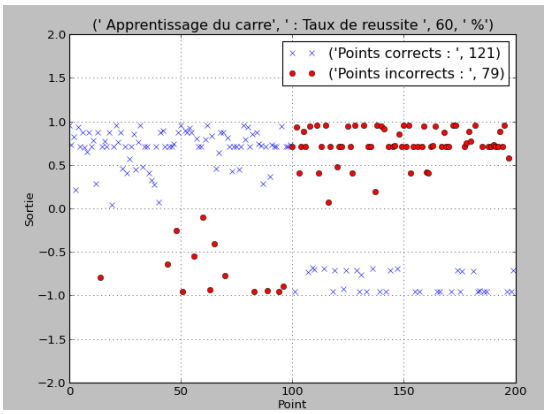
A ce niveau nous avons remarqué que la forme qui apprend le moins bien est le carré et la mieux apprise est le triangle.

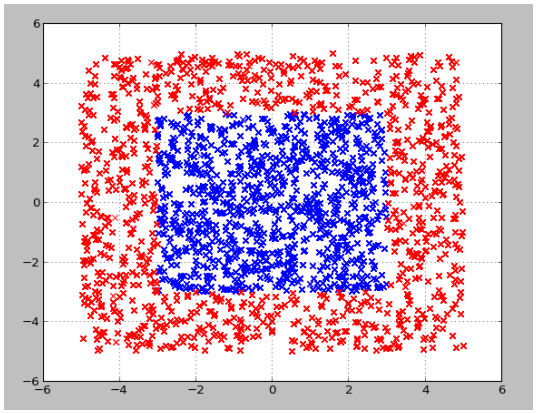
Nous avons alors supposé l'importance du style et de la taille de la forme. Plus la forme sera grande et complexes, plus il faudra de samples pour la caractériser. De plus, il faut savoir que plus il y aura de samples, plus il faudra de cycles afin d'atteindre un apprentissage optimum du perceptron.

Pour vérifier cette hypothèse nous avons modifié la taille du carré sans modifier le nombre de samples et de cycles :









On remarque bien une grande différence d'apprentissage entre les différentes tailles.

On arrive donc à la limite de notre expérience vu que lors de la création des formes, nous n'avons pas pris en compte leurs tailles. Ce paramètre biaise donc nos résultats.

Conclusion

Le perceptron est donc un moyen rapide est efficace pour réaliser un apprentissage simple. Plus l'objet à apprendre sera complexe, plus il faudra de neurones et donc de puissance de calculs pour réaliser l'apprentissage. Lorsque la forme tend vers un polygone possédant un nombre infini de côté comme le cercle, il est nécessaire de créer un perceptron avec un nombre illimité de neurones.

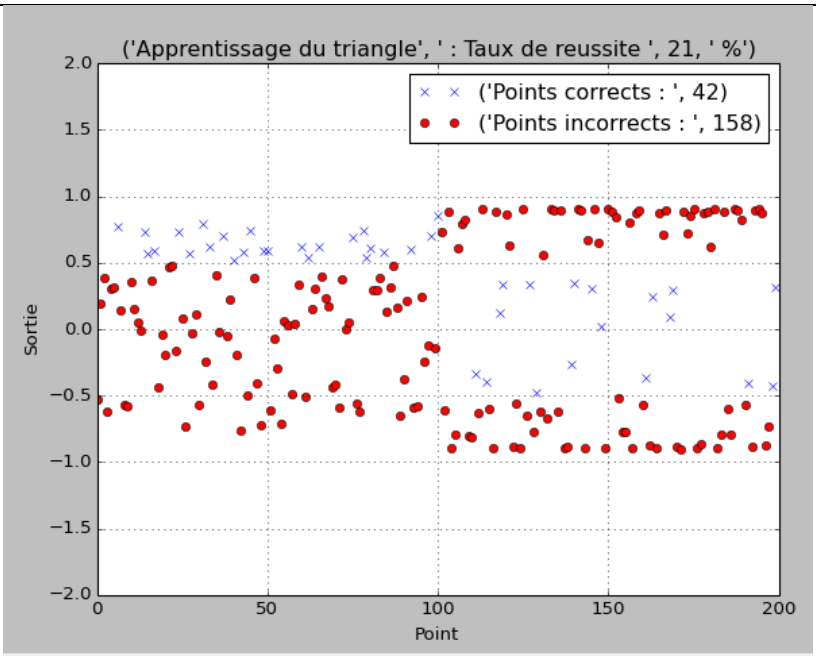
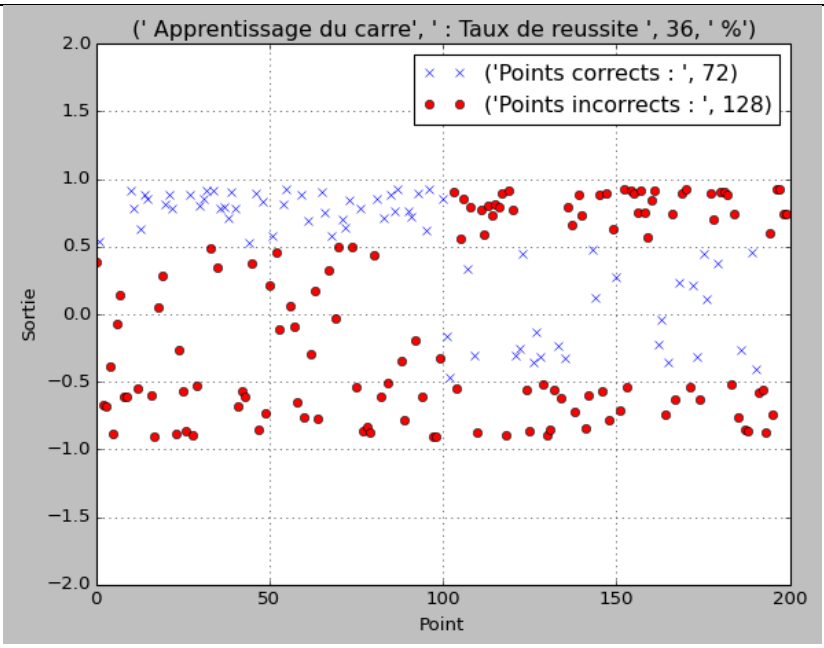
C'est pourquoi la limite principale du perceptron est sa capacité restreinte à apprendre des formes complexes ainsi que le nombre de samples nécessaire à l'apprentissage d'une forme de grande taille.

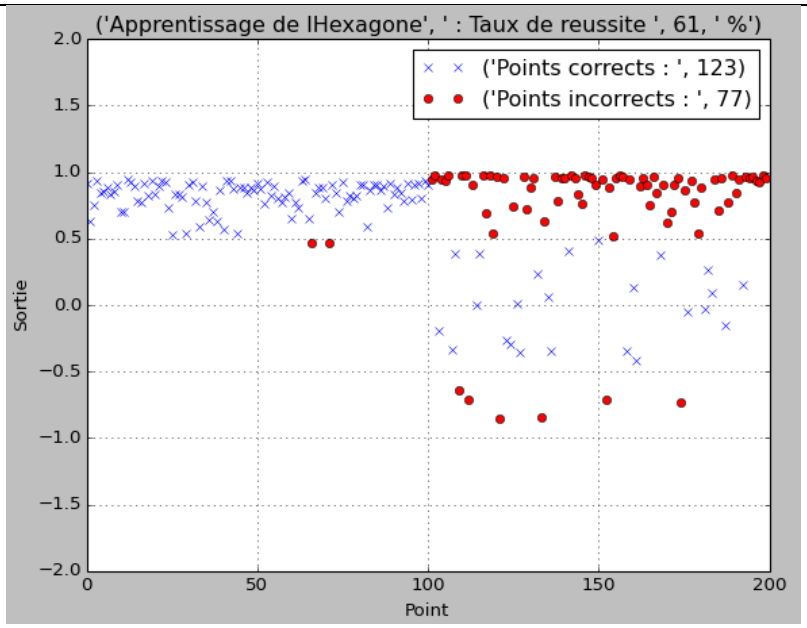
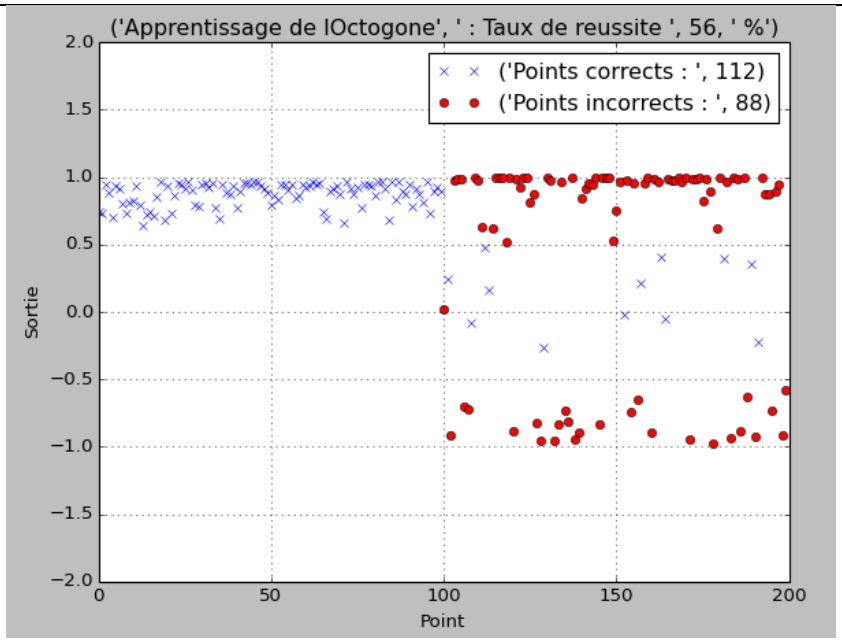
Cependant le perceptron ne calcule pas suivant la taille de la forme. En effet chacun de ses neurones de la couche cachée apprend en réalité les équations des droites caractérisant chaque côté. C'est pourquoi il serait plus intelligent d'apprendre au perceptron un « modèle réduit » de la forme en laissant un scalaire en paramètre afin de pouvoir changer la taille de la forme sans avoir à refaire un apprentissage plus exhaustif.

Annexes : capture écran du rendu de l'application

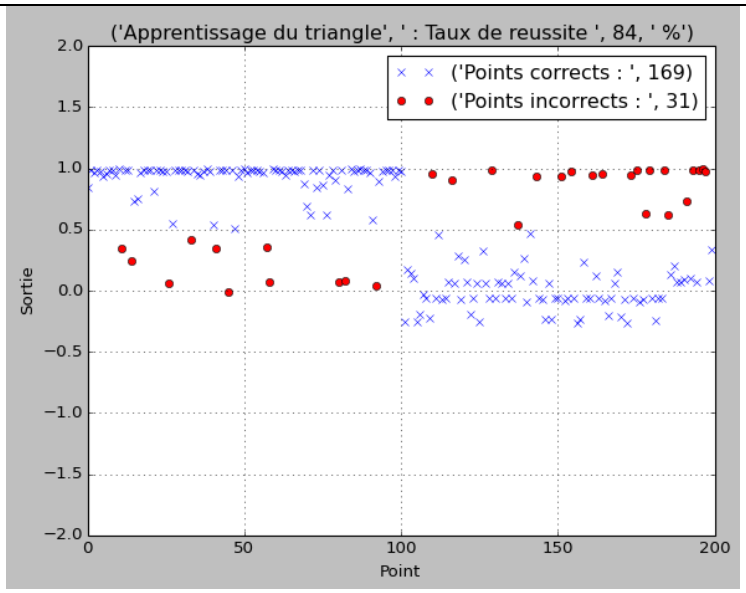
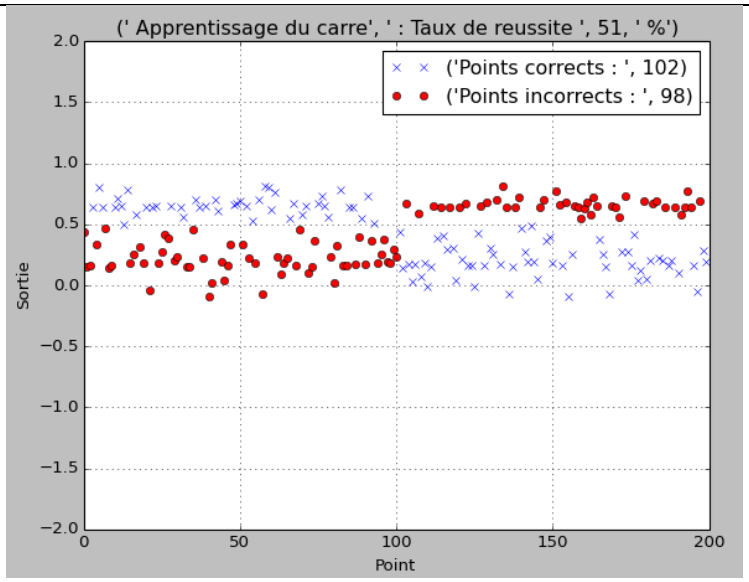
Avec en sortie 0 et 1 :

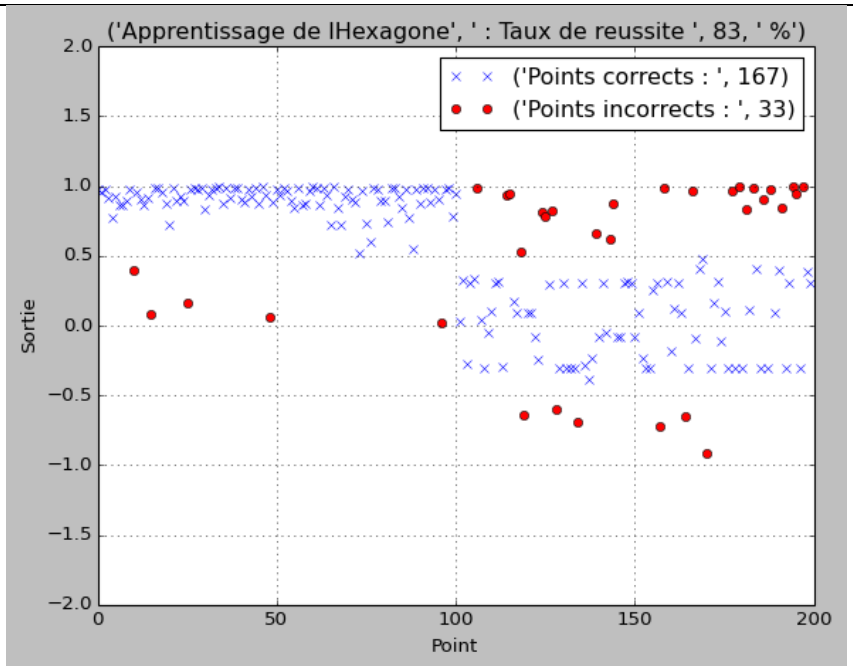
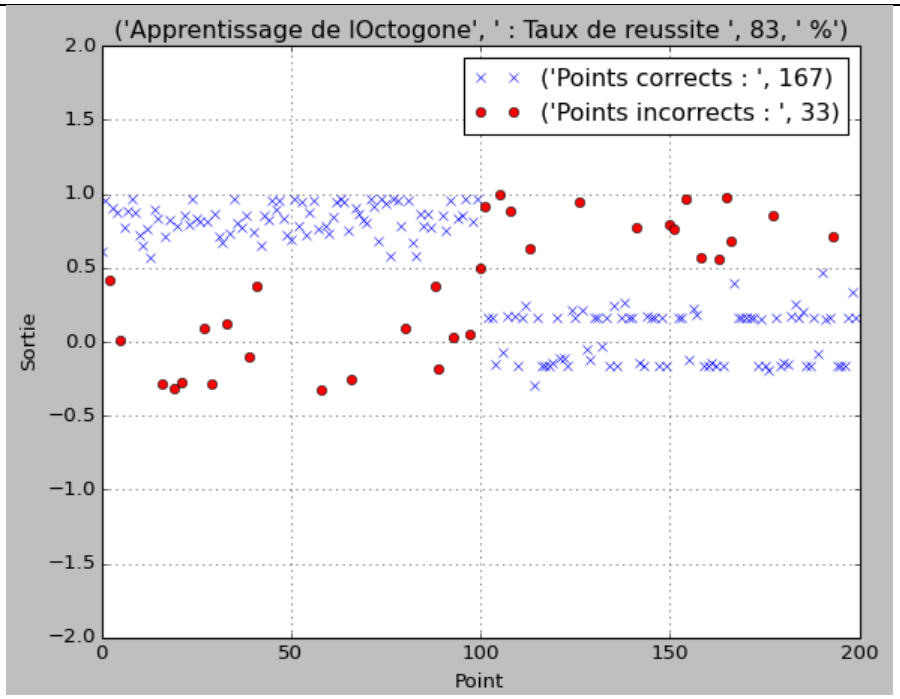
Cycle=2 et nombre de samples = 100

Triangle :	
Carré	

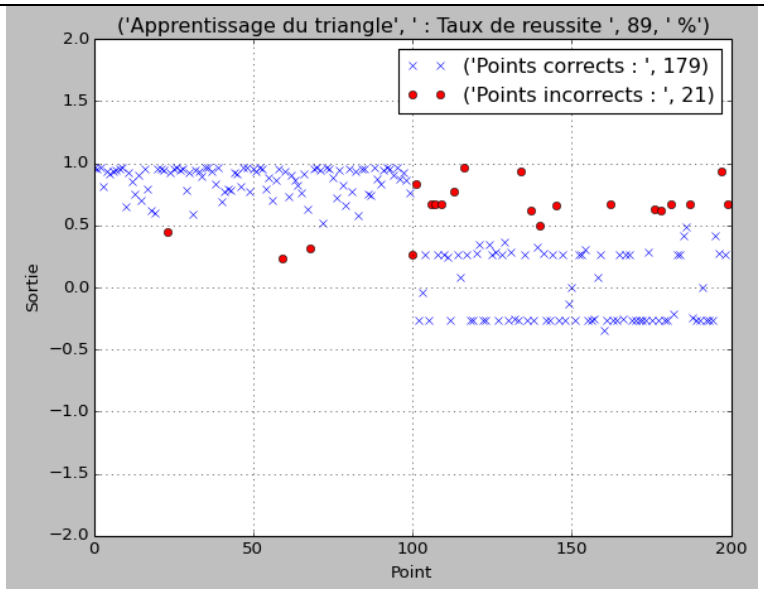
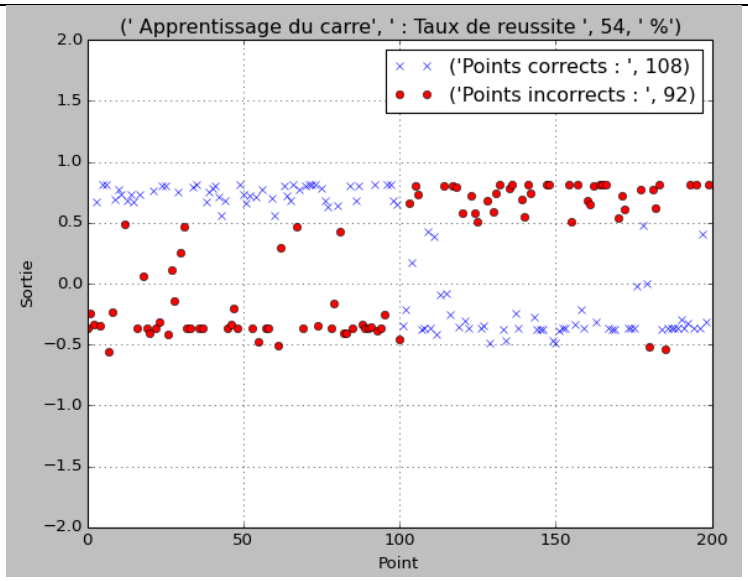
Hexagone	
Octogone	

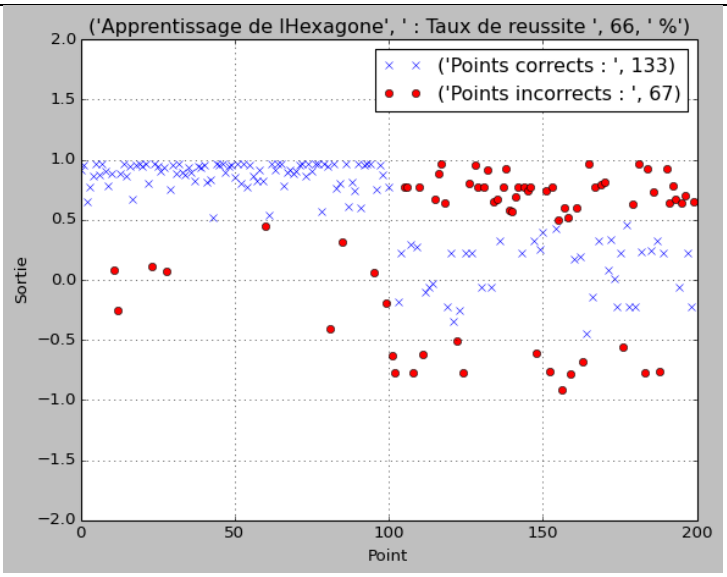
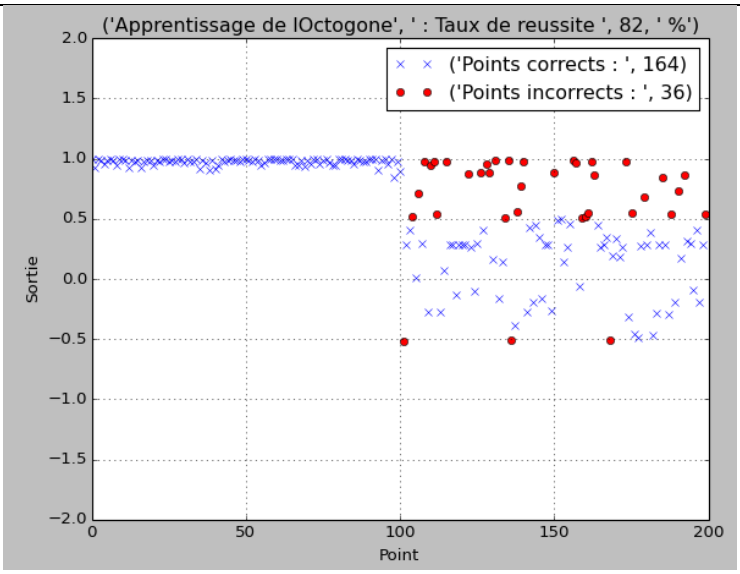
Cycle=2500 et nombre de samples = 100

Triangle :	
Carré	

Hexagone	
Octogone	

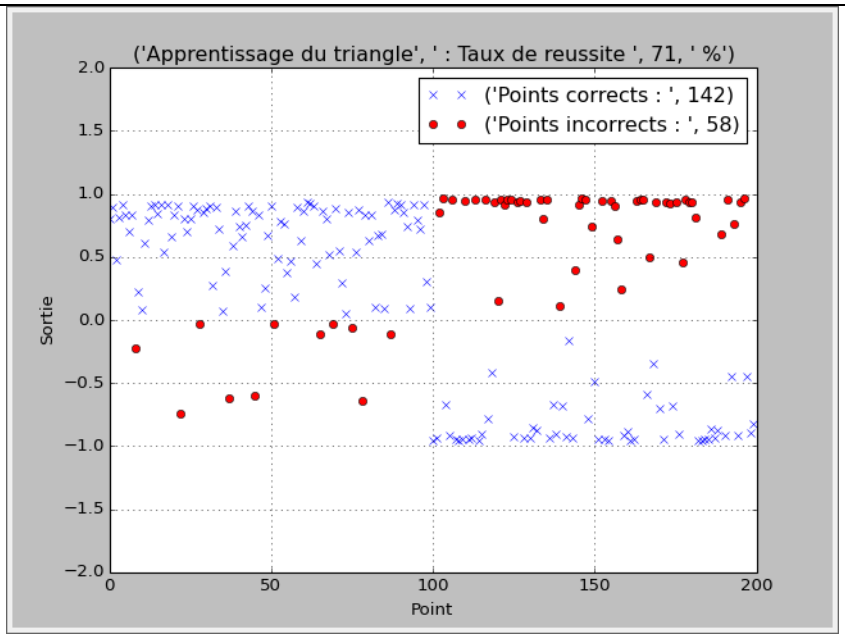
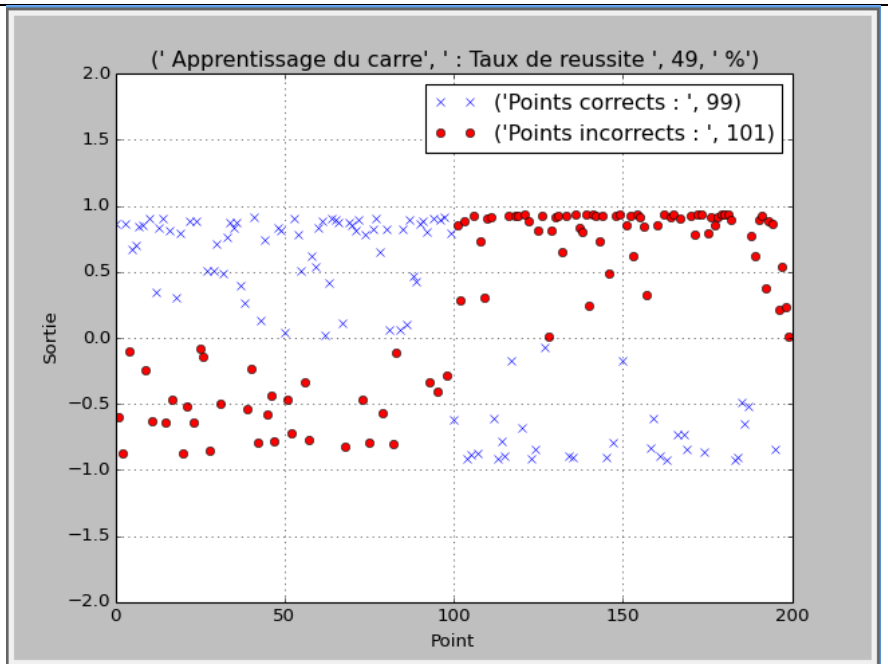
Cycle=2500 et nombre de samples = 2000

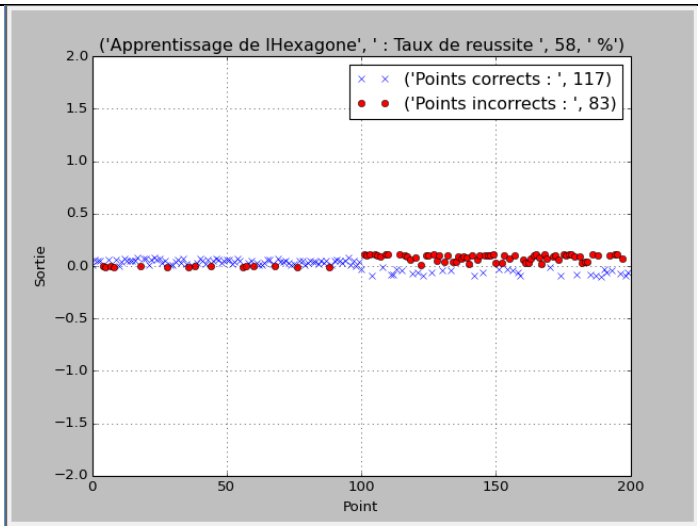
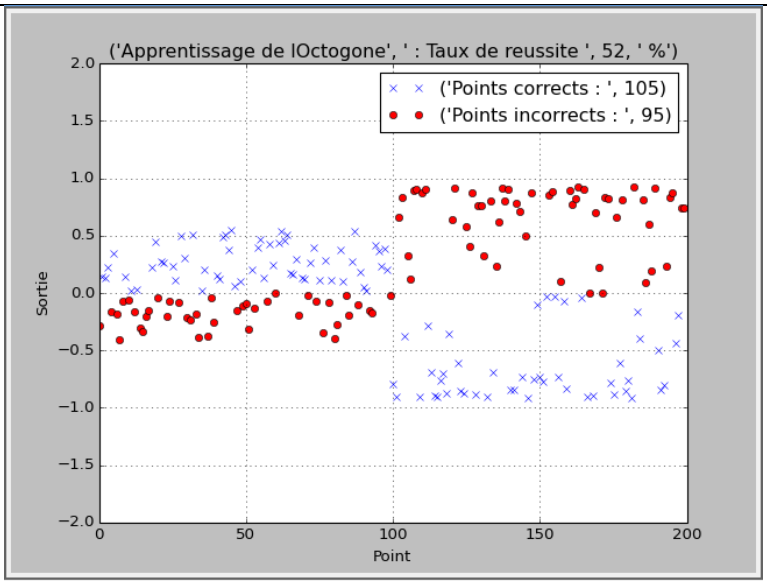
Triangle :	
Carré	

Hexagone	
Octogone	

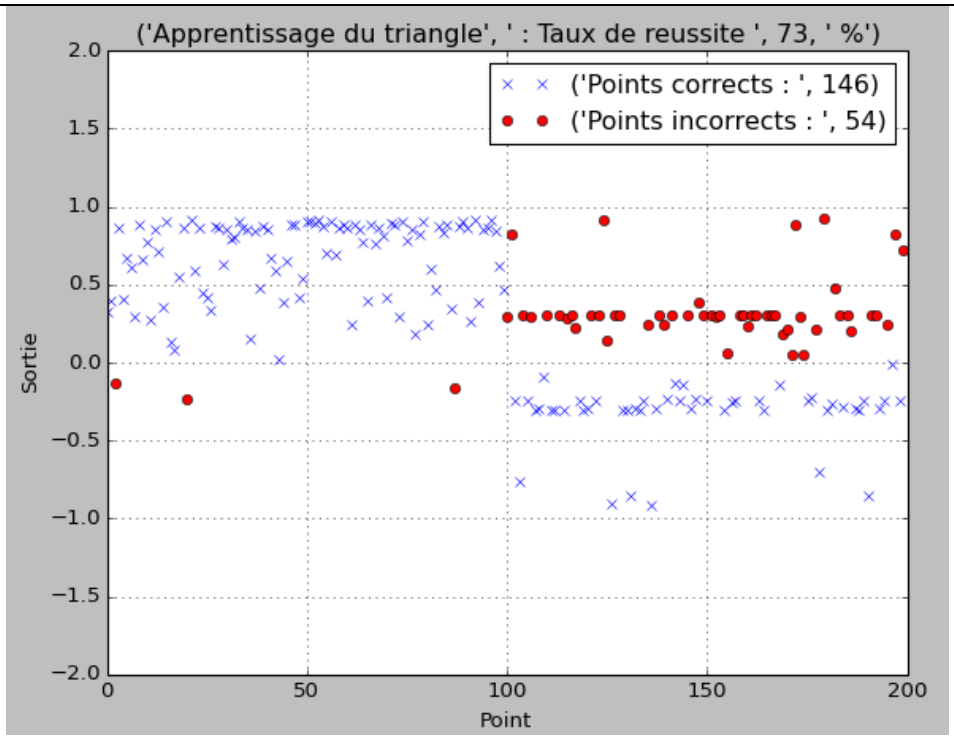
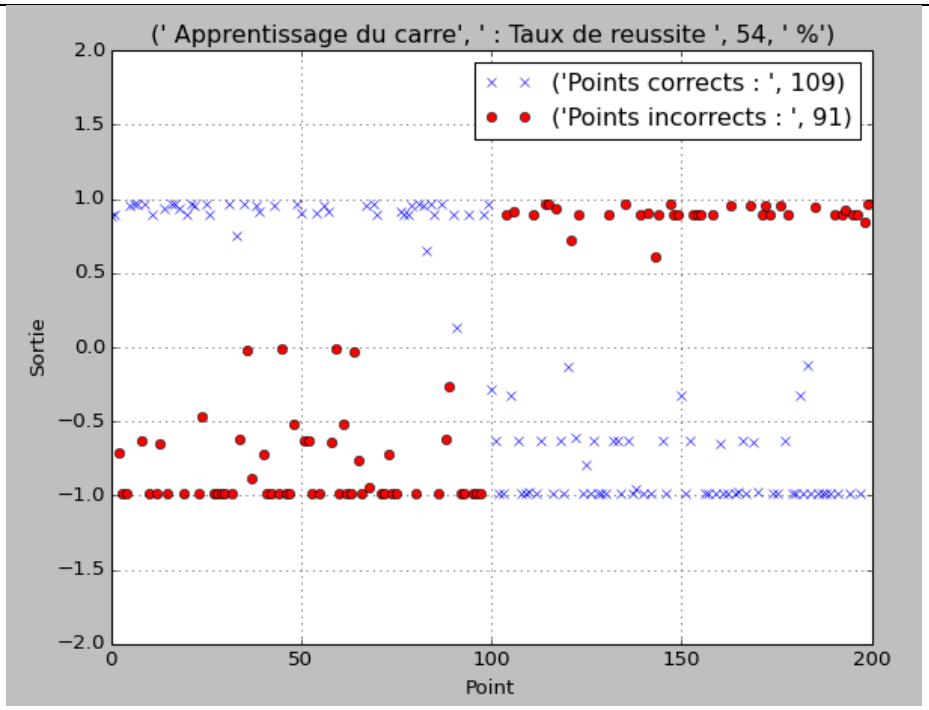
Avec en sortie 1 et -1 :

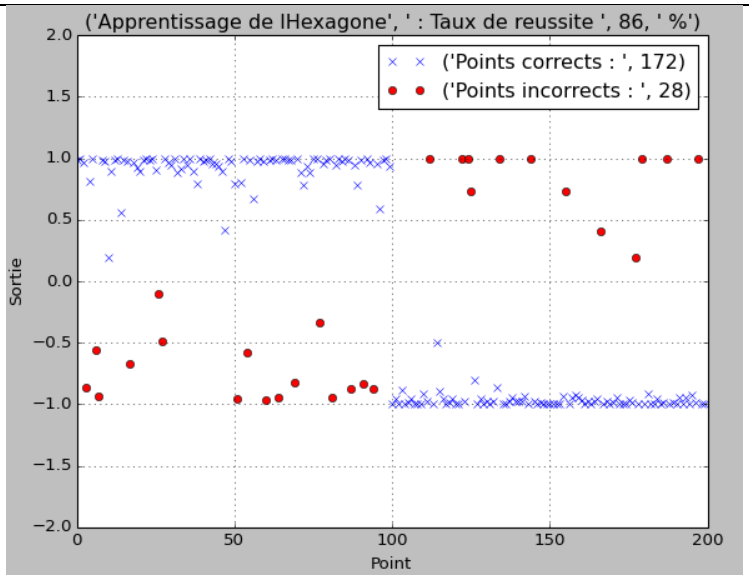
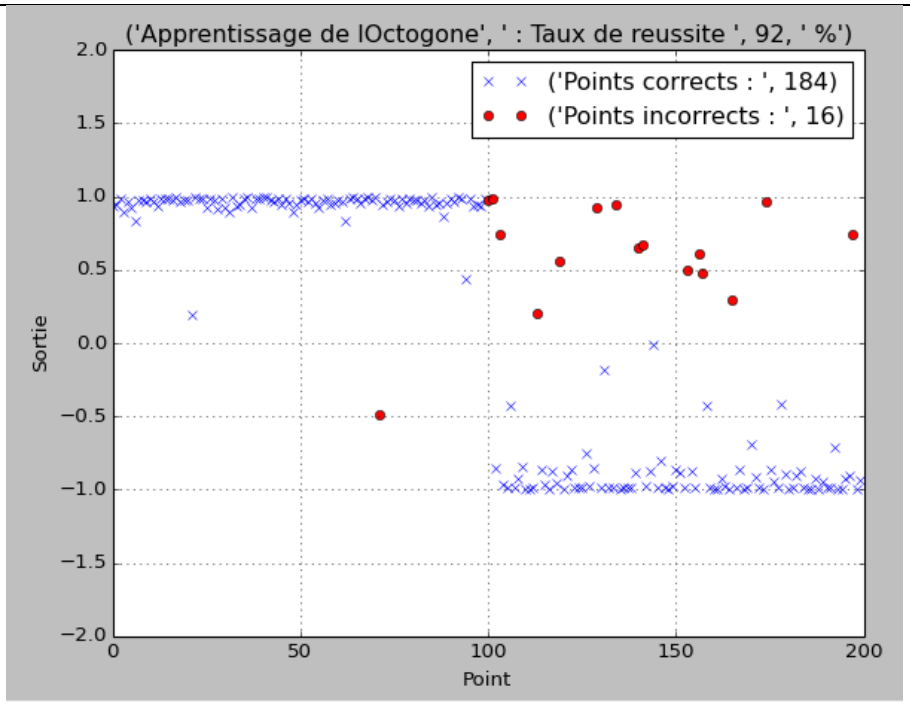
Cycle=2 et nombre de samples = 100

Triangle :	
Carré	

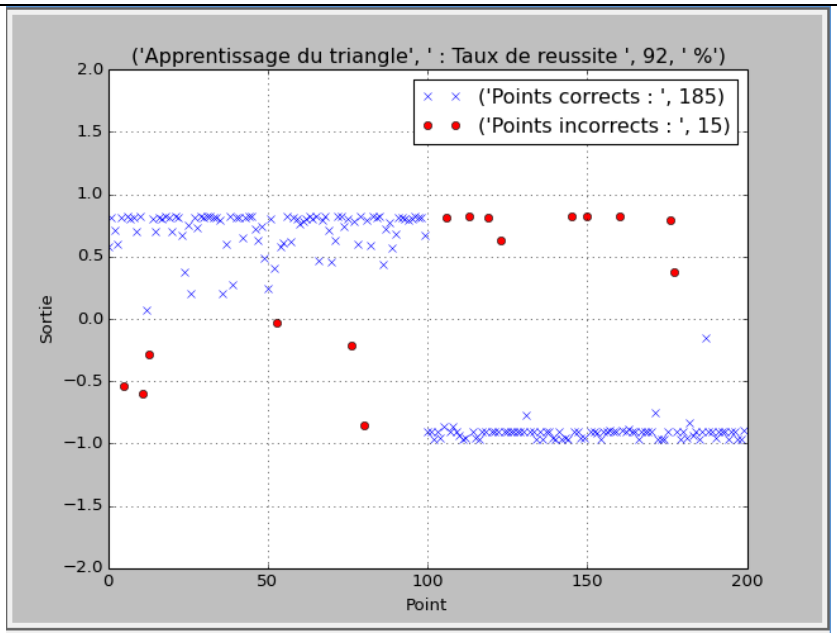
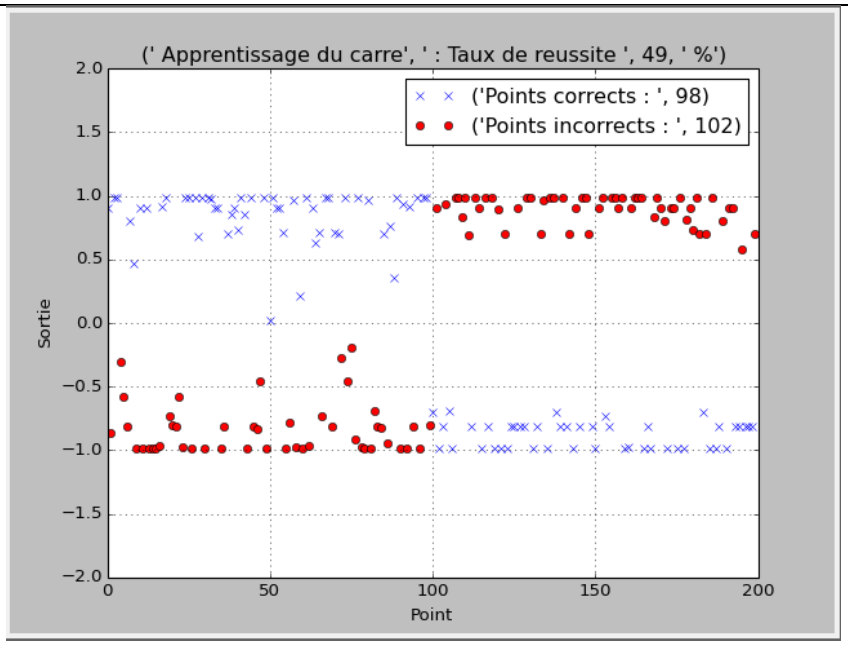
Hexagone	
Octogone	

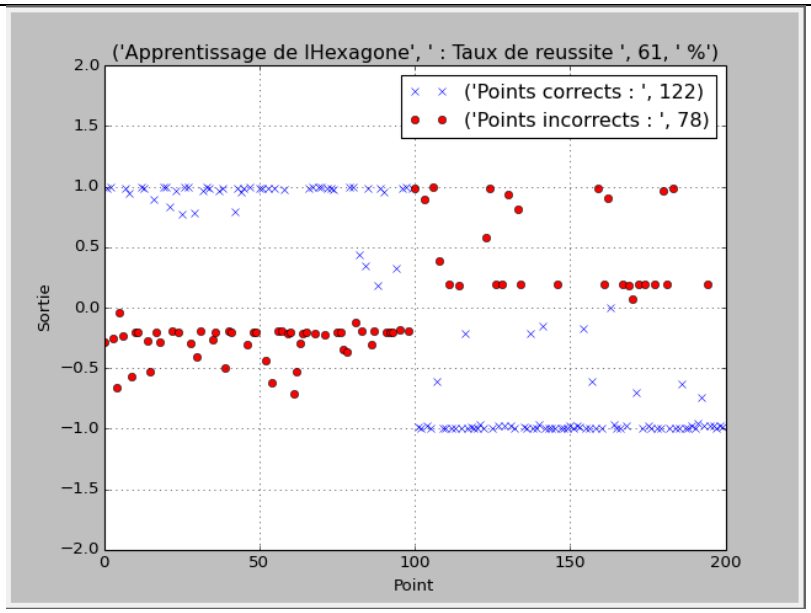
Cycle=2500 et nombre de samples = 100

Triangle :	
Carré	

Hexagone	
Octogone	

Cycle=2500 et nombre de samples = 2000

Triangle :	
Carré	

Hexagone	
Octogone