Mémoire de fin d'études

Les réseaux de neurones dans les jeux vidéo

Quelle est l'utilisation de l'apprentissage numérique dans les jeux vidéo ?

Fabien SACRISTE 5 AIJV
Guillaume NOISETTE 5 AIJV
Maître de Mémoire : Nicolas VIDAL

ESGI école supérieure de genie informatique

5A 2014

RÉSEAU GES GRANDES ÉCOLES SPÉCIALISÉES

Table des matières

Résumé

Ce mémoire traite de l'utilisation de l'apprentissage numérique dans les jeux vidéo. Nous avons décrit dans un premier temps les différents types d'apprentissage que l'on peut distinguer, et dans quels jeux nous pouvons les retrouver. Dans un deuxième temps, nous avons mis en place un perceptron multi-couches afin de reconnaître différents comportements de patrouille. Ce projet nous a permis de comprendre pourquoi les jeux vidéo implémentent si peu les algorithmes d'apprentissage numérique. En effet, leur complexité d'implémentation et leur temps de paramétrage extrêmement long rendent cette tâche fastidieuse et contraignante. Grâce à ce mémoire, nous avons vu les difficultés de l'auto-formation face à un sujet totalement inconnu. De plus, nous avons pu comprendre le fonctionnement précis du perceptron multi-couches en l'appliquant à un contexte lié au jeu vidéo.

Abstract

This master's thesis is about the use of machine learning in video games. In order to introduce the topic, we described the different kinds of learning that exist and in which games they are applied. For the second step, we used a multi-layers perceptron to recognize different patrol movements. This project helped us to understand why machine-learning algorithms don't feature a lot in video games. Indeed, the complexity of their implementation and the large amount of time it takes to adjust the settings make this task tedious and binding. Thanks to this thesis, we faced the difficulty of self-learning whilst dealing with an unknown subject. In addition, this project made us understand precisely how the multi-layers perceptron works by applying it to the context of a video game.

Mots clés

Jeux vidéo
Apprentissage numérique
Mode supervisé
Carte de Kohonen
Classifieur Linéaire
Intelligence artificielle
Réseau de neurones
Perceptron
Arbre décisionnel
Reconnaissance de forme

Key words

Video games
Machine learning
Supervised mode
Kohonen map
Linear classifier
Artificial intelligence
Neuronal network
Perceptron
Decisional tree
Shape recognition

Remerciements

Tout d'abord, nous tenons à remercier, Monsieur Vidal, notre maître de mémoire pour ses conseils techniques pour l'orientation et l'élaboration de notre projet, mais également pour sa précieuse aide tout au long de notre cursus.

Nous souhaitons également remercier Monsieur Lioret notre professeur principal, pour ses conseils et ses nombreuses orientations de recherches ainsi que son support dans l'ensemble de nos projets.

Nous tenons également à témoigner toute notre gratitude à l'ensemble du corps enseignant et à Monsieur Peter, directeur de l'ESGI, pour leur soutien, leur aide, leur patience et bien sûr le partage de leurs connaissances.

Pour conclure, nous remercions plus particulièrement nos tuteurs pour leur aide au quotidien, leur support ainsi que leur patience dont ils ont fait preuve durant ces années d'études.

Introduction

C'est dans les années 1970 que les jeux vidéo ont commencé à faire leur apparition. Développés sur des machines peu puissantes, les programmes se devaient de ne pas être trop compliqués et étaient limités en espace physique.

Au fur et à mesure que le jeu vidéo a évolué, les technologies algorithmiques et matérielles ont elles aussi avancées, ouvrant la porte aux développeurs pour plus d'audace et d'innovation.

Les recherches en matière d'apprentissage numérique étant relativement poussées, il était tout naturel pour les studios de développement de commencer à les implémenter dans leurs projets pour ajouter plus de réalisme et une meilleure expérience de jeu.

Ce mémoire a pour but dans un premier temps, de mettre en valeur les algorithmes d'apprentissage numérique les plus répandus et utilisés dans l'univers vidéoludique ainsi que de mettre en avant quelques jeux ayant utilisés ces technologies.

Dans un second temps, nous étudierons un programme développé exclusivement dans le cadre de ce mémoire, afin d'étudier un algorithme du perceptron multi-couches (PMC) utilisé pour la reconnaissance de forme. Un total de 10 formes allant de la plus simple (ligne droite) à la plus compliquée (forme en étoile) devront pouvoir être reconnues par notre programme. Chaque forme représentera le mouvement d'une patrouille d'une unité dans un jeu.

Cette étude servira à mettre en avant l'exploitation du PMC dans un cadre bien précis, et les différentes variables constituant le programme influant directement sur l'apprentissage et la résolution des formes. Nous approfondirons les résultats obtenus afin de faire ressortir les différentes configurations de paramétrages ainsi que leur points forts et points faibles.

Apprentissage numérique

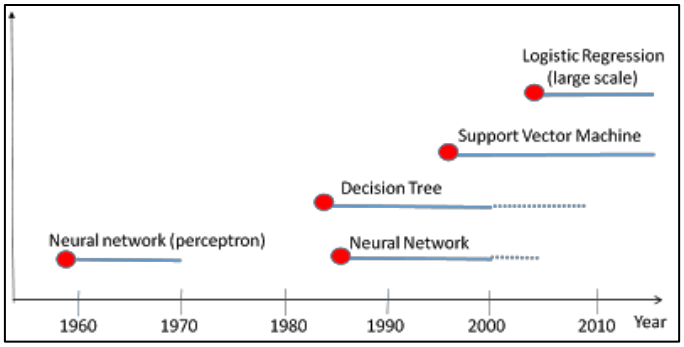
Définition

L'apprentissage dit numérique (ou automatique) est un apprentissage qui est réalisé par un ordinateur (ou du moins une machine) à partir de différentes données qui lui sont transmises en entrée. Le but principal de cette méthode est de passer outre les limites d'algorithmes classiques par exemple des arbres de décisions qui peuvent être très compliqués pour la classification de données, on parle alors d'explosion combinatoire. Le traitement des entrées de ce système est réalisé à l'aide de différents algorithmes suivant la méthode d'apprentissage qu'on lui impose, mais il est toujours basé sur de la probabilité ce qui induit une possibilité d'erreur de sortie.

Il existe plusieurs modes d'apprentissage : le supervisé, le non supervisé et le semi-supervisé (appelé aussi hybride). Ces différents modes seront abordés plus loin dans ce mémoire.

Historique

Arthur Samuel a été un des premiers à créer un programme permettant de jouer au jeu de dames pour IBM. Sa première version fonctionnelle fut délivrée en 1955 et était capable d'apprendre en jouant contre une autre personne. La technique d'apprentissage supervisée utilisée à l'époque, est celle qu'on appelle plus communément aujourd'hui la méthode de recherche heuristique.



Après cette première avancée, de nombreuses découvertes ont été faites, notamment dans les réseaux de neurones avec le Perceptron en 1957 (abordé plus

loin dans ce mémoire). Le système ELIZA développé par Joseph Weissenbaum en 1966, est un programme permettant de simuler un psychothérapeute. Evidemment les échanges sont très limités, et le but n'a jamais été de remplacer une véritable personne. Puis, après une longue période sans grandes avancées, ce n'est que dans les années 1980 que de nombreuses découvertes ont été faites, notamment dans le domaine de l'intelligence artificielle. On a ainsi pu voir apparaître le système d'arbre décisionnel et les réseaux de neurones multi-couches très répandus dans le cadre de la finance ou d'autres secteurs d'activités comme les systèmes postaux pour la reconnaissance des adresses. D'autres découvertes ont eu lieu jusque dans les années 2000, notamment grâce à l'arrivée d'internet ouvrant ainsi la porte à d'infinies quantités d'informations.

Présence actuelle dans les jeux vidéo

L'apprentissage numérique a commencé à apparaître dans les jeux vidéo en 1996 avec les réseaux de neurones dans *Battlecruiser 3000AD* et dans *Creatures*, puis en 2000 avec *Colin McRae Rally 2.0* (PS1/Dreamcast) et 2001 dans *Black & White*. Au fur et à mesure des années, les réseaux de neurones se sont développés et ont commencé à être implémentés dans divers domaines tel que dans la reconnaissance d'objets ou de caractères, les moteurs de recherches, le milieu médical, le secteur boursier, etc… Si l'apprentissage numérique s'est si bien développé ces dernières années dans de nombreux secteurs, il n'a cependant pas spécialement gagné sa place dans le monde du jeu vidéo. En effet, on dénombre seulement quelques grosses licences tels que *Forza Motorsport* (2005) sur Xbox qui avait la capacité d'apprendre à partir de la conduite du/des joueur(s), ou bien *Halo 3* (2007) sur Xbox 360 pour la gestion des matchmaking sur internet en fonction des différents skills et statistiques des joueurs. A une échelle moins grande, quelques petits jeux tels que *Galactic Arms Race* ont implémenté la technologie des réseaux de neurones (cgNEAT) afin de créer de manière complètement indépendante des armes pour un vaisseau évoluant dans l'espace. A l'aide d'indication du joueur, le jeu génère les nouvelles armes selon que les anciennes lui ont plu ou non. Dans un autre registre des algorithmes ont été mis en place pour pouvoir résoudre des jeux

de logique tels que les échecs ou le jeu de Go, un ancien jeu chinois dont une version digitale est sortie sur le XBLA en 2010.

Différents modes d'apprentissage

Dans le monde de l'apprentissage numérique, 3 modes se distinguent : le supervisé, le non-supervisé et l'hybride (une combinaison des 2). Ces modes ont des caractéristiques différentes de fonctionnement, et permettent de regrouper l'ensemble des algorithmes et modèles d'apprentissage.

Notre étude se basant sur les jeux vidéo, seul le mode supervisé sera développé. En effet, les 2 autres modes ne sont que très peu utilisés, voire pas du tout par les développeurs faisant partie de studios.

Mode supervisé

Le but du mode supervisé est d'amener l'ordinateur à apprendre un système de classification fourni au préalable. Celui-ci consiste à apprendre en minimisant le nombre d'erreurs tout en respectant les entrées données. Les deux techniques les plus exploitées utilisant ce système, sont les réseaux de neurones et les arbres de décisions. Dans le cadre de la première, la classification est utilisée afin de déterminer le nombre d'erreurs du réseau puis de le modifier afin d'en réduire le nombre, alors que l'arbre décisionnel, lui, crée un modèle capable par la suite de prédire la valeur d'une variable cible basée sur les entrées reçues.

Les réseaux de neurones artificiels

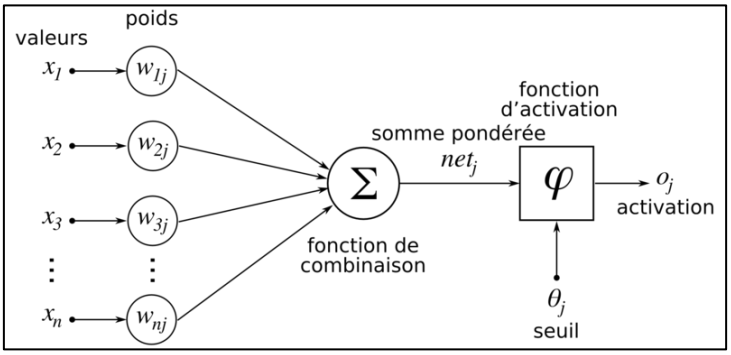
Définition

Un réseau de neurone artificiel (RNA) est une façon de traiter l'information calquée sur le fonctionnement du système nerveux biologique, comme le cerveau. De la même manière que le cerveau humain utilise un système de réseaux inter-connectés pour résoudre des problèmes, un RNA est composé de nombreuses fonctions reliées entre elles, prenant des données en entrée, et renvoyant une ou

plusieurs informations en sortie. Toujours sur le même schéma que le cerveau humain, un RNA a besoin d'être entraîné et doit subir des ajustements afin de répondre le mieux à une situation/problème donné(e).

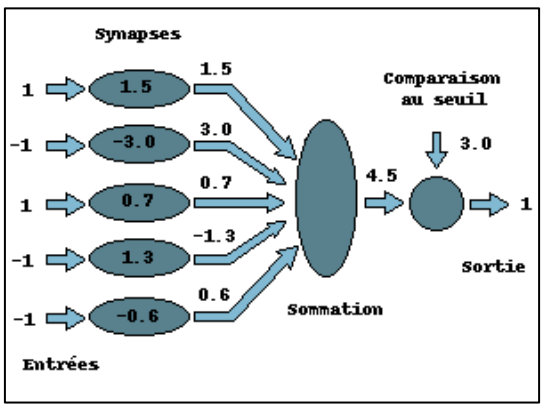
Fonctionnement

Comme expliqué précédemment, un RNA fonctionne grâce à l'inter-connexion de différentes fonctions. Ces connexions sont représentées sous formes de couches, chaque couche est composée d'un certain nombre de neurones recevant chacun des informations en entrées provenant des neurones de la couche précédente.



D'une couche à l'autre, les neurones sont tous reliés entre eux par des synapses qui sont elles-mêmes pondérées d'un poids différent selon l'importance du neurone. La sortie du neurone de la couche précédente est multipliée par ce poids et est ensuite sommée à l'ensemble des autres neurones multipliés par le poids de leur synapse. Arrivée à la dernière couche, la sortie du/des neurone(s) est/sont alors comparé(s) à une valeur seuil, et selon le résultat de la comparaison une décision ou un résultat sera alors renvoyé.

Exemple :



Dans le monde du jeu vidéo

Les réseaux de neurones peuvent être utilisés de deux manières différentes dans les jeux vidéo. La première consiste à les utiliser au moment du développement du jeu; ils apprendront à partir de données fournies par les développeurs, puis le résultat final sera inclus dans le jeu. Les développeurs de Codemasters ont utilisé ce processus dans le jeu *Colin McRae Rally 2.0* sorti en 2000 afin d'entraîner une IA permettant à la voiture de rester sur la route, d'avoir un comportement cohérent, mais surtout de pouvoir s'adapter à tous types de circuits, sans avoir été entraînée dessus au préalable.

La deuxième technique, consiste à inclure le RNA directement dans la version finale du jeu, de cette manière, le jeu apprendra à partir du comportement du joueur, et les informations collectées seront alors restituées en jeu. Si cette technique est coûteuse en ressources, avec l'avancée des technologies et des supports, elle est désormais moins compliquée à implémenter dans les jeux. De nos jours, de nombreux jeux l'implémente afin d'offrir une meilleure expérience au joueur, de ce fait il y a moins d'effet de redondance et de linéarité car les PNJs (Personnage Non Joueur) sont capables de s'adapter aux routines du joueur, et par conséquent de l'obliger à changer ses habitudes. L'un des premiers jeux à l'avoir mis en oeuvre est *Creatures* en 1996. Le principe du RNA était de permettre une adaptation du comportement de la créature en fonction des ordres et actions du joueur. Au fur et mesure que le joueur avançait dans le jeu, le comportement et les réactions de la créature évoluaient, ainsi, d'un joueur à l'autre, des créatures pouvaient avoir des agissements complètement différents. En 2001 Lionhead sort *Black and White*, le but du jeu étant là aussi de contrôler une créature, mais en symbolisant une divinité sur Terre. De la même manière que *Creatures*, le personnage contrôlé s'adapte et change de comportement en fonction de l'éducation que le joueur lui inculque.

Avec l'avancée des technologies, les réseaux de neurones ont énormément évolué ainsi que la manière dont les développeurs s'en servent. En 2005, Microsoft sort le jeu de course *Forza Motorsport* et y inclut sa technologie développée pour l'occasion : *Drivatar*. Celle-ci est capable d'analyser le comportement du joueur et de s'y adapter, mais également de définir le chemin le plus optimisé pour parcourir un circuit puis d'altérer sensiblement ce chemin afin d'obtenir un comportement plus "aléatoire", donc plus proche de celui d'un humain. La récente sortie de la Xbox One (le 22 novembre 2013) ainsi que sa capacité à se connecter dans le cloud, a permis

aux développeurs de pousser un pas plus loin l'utilisation de *Drivatar* dans leur dernier jeu *Forza Motorsport 5*. En effet, en plus des possibilités citées précédemment, Drivatar est capable d'analyser et de calquer le style de conduite du joueur. Grâce au cloud, le profil de conduite du joueur est alors disponible dans le monde entier. La résultante et le but de cette prouesse, est d'avoir la sensation de jouer contre son ami sur internet, alors qu'il ne s'agit en fait que d'un programme restituant son style de jeu.

Point négatif

Comme pour tout algorithme d'apprentissage, il y a un danger; celui du surapprentisage. Cela arrive lorsque l'algorithme apprend trop, et par conséquent va par la suite trop restreindre son champ d'action. L'impact majeur est que le résultat obtenu n'est plus du tout conforme aux attentes.

En 2011, un développeur décide de mettre en place un RNA pour générer de bonnes IA pour les bots du jeu *Quake III*. Afin d'obtenir un comportement basé seulement sur le comportement des bots, il en place 16 dans une arène qui interagiront uniquement entre eux, sans intervention humaine, le tout placé sur un serveur à lui. Avec le temps, il oublie qu'il a lancé son expérience, et laisse fonctionner son programme pendant 4 ans. Lorsqu'il se rappelle de son expérience et regarde les résultats, il s'attend à avoir des bots avec un comportement très réalistes et performants. Au lieu de ça, il constate qu'aucun des bots ne bougent ni attaquent les autres adversaires. Cela s'explique par le fait que l'algorithme a subi un surapprentissage, et à déterminer que le meilleur moyen de gagner était celui de ne pas attaquer, ni même de bouger.

Cet échec démontre bien la limite des RNA, et qu'il faut bien déterminer à quel moment l'algorithme a atteint le meilleur de son "raisonnement".

Les arbres décisionnels

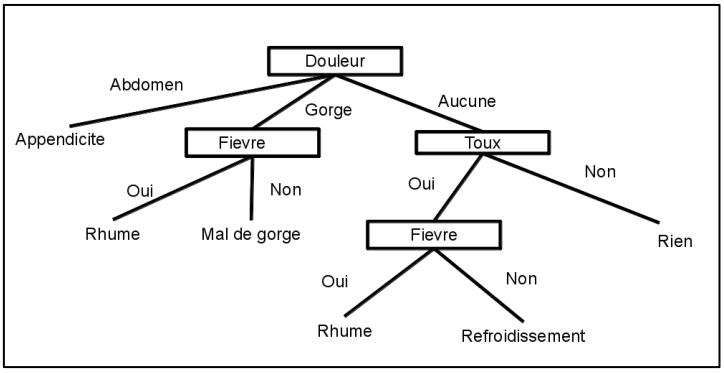
Définition

Un arbre décisionnel est un graphe représentant une structure composée de décisions et d'actions reliées par des branches sur différents niveaux. Le but d'un

arbre de décisions est de pouvoir mettre en place un système simple et clair aidant à faire des choix selon une situation donnée.

Fonctionnement

L'arbre de décision commence généralement avec un choix à faire. Chaque choix est représenté par une branche menant elle-même à une autre décision à prendre, appelée plus communément noeud. Ainsi, un arbre est la représentation d'une succession de noeuds reliés par des branches. Si un noeud n'a pas de branche, il est alors appelé feuille et représente la décision finale de l'arbre.



Dans un arbre décisionnel classique, un noeud peut avoir autant de branches que nécessaires, cependant il existe un autre type d'arbre, dit binaire, qui lui ne peut avoir qu'au maximum 2 branches.

Les points positifs d'un arbre de décision, sont qu'ils sont faciles à implémenter, rapides d'exécution et simples de compréhension.

Voici un pseudo code représentant l'image d'exemple ci-dessus :

```
Si douleur = abdomen alors
        problèmeSanté = Appendice
Sinon Si douleur = Gorge alors
        Si Fièvre = Vrai alors
                problèmeSanté = Rhume
        Sinon
                ProblèmeSanté = MalDeGorge
        FinSi
Sinon
        Si Toux = Vrai alors
                Si Fievre = Vrai alors
                        problèmeSanté = Rhume
```

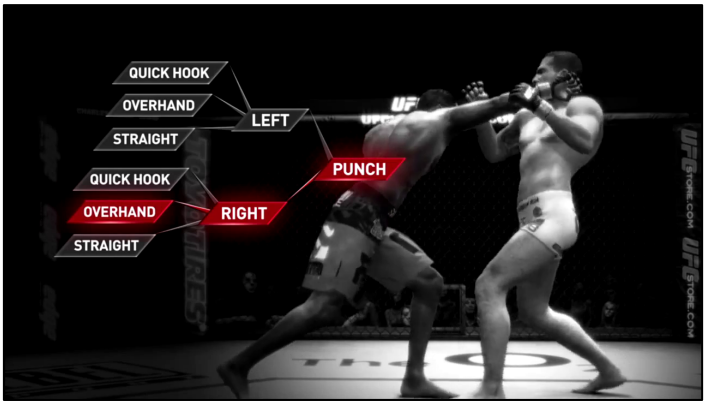
```
            Sinon
                problèmeSanté  = Refroidissement
            FinSi
        Sinon
            problèmeSanté = Aucun
        FinSi
FinSi

Renvoyer problèmeSanté
```

Dans le monde du jeu vidéo

L'arbre de décisions n'est pas la technologie la plus répandue dans l'univers du jeu vidéo. Cependant elle a longtemps été utilisée dans les jeux de stratégie (STR) afin de faire du *path finding*. Même si les résultats pouvaient parfois être douteux (ex: une armée entière essayant de passer entre deux murs très rapprochés). Cependant, cette technique suffisait à remplir tant bien que mal ses tâches. Depuis, d'autres algorithmes tels que A* ou Djikstra ont pris le relais dans ce domaine.

C'est en 2012 que THQ sort le jeu *UFC Undisputed 3*. Le système de combat du jeu est basé sur des arbres décisionnels déclenchés selon certaines situations. Par exemple, si un des combattants ne cesse de se protéger au visage, l'adversaire va le percevoir, et parcourir un de ses arbres pour voir quelle action est la plus adaptée, ce qui pourrait être d'attaquer au niveau du ventre.



Dans un autre registre, c'est en 2007 qu'est apparu le site web Akinator, un génie sorti tout droit de sa lampe pour nous dire à qui on pense. Le système est simple, il suffit que le joueur pense à quelqu'un et réponde à une série de questions concernant le personnage. Les questions sont toutes de types fermés, c'est-à-dire

qu'on peut seulement répondre par oui ou non. L'architecture de l'algorithme est gardée secrète par le développeur, Arnaud Megret, cependant on sait que le système est basé sur un arbre décisionnel binaire. De ce fait, pour chaque question posée par l'ordinateur, la réponse donnée par le joueur élimine un lot de possibilités, et lorsque l'algorithme "pense", avoir trouvé la bonne personne, il l'envoie au joueur. Il est cependant possible qu'une mauvaise personne soit soumise au joueur, soit parce qu'Akinator ne connait pas la personne en question, ou bien parce que le joueur a répondu n'importe quoi. Dans le cas de la première hypothèse, l'algorithme est capable d'apprendre le personnage en demandant simplement la bonne réponse au joueur, ainsi avec la succession de réponses fournies par le joueur, le programme sera en mesure d'apprendre les caractéristiques du personnage. Ce système bien que bluffant lorsqu'on y joue, repose sur un système simple, où le parcours d'un simple arbre suffit à déterminer la meilleure solution, l'élément clé de cette application étant bien sur la base de données et la capacité d'apprentissage.

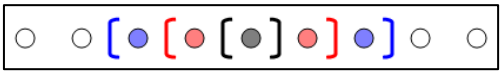
Acquisition des données par les réseaux de neurones

Carte de Kohonen

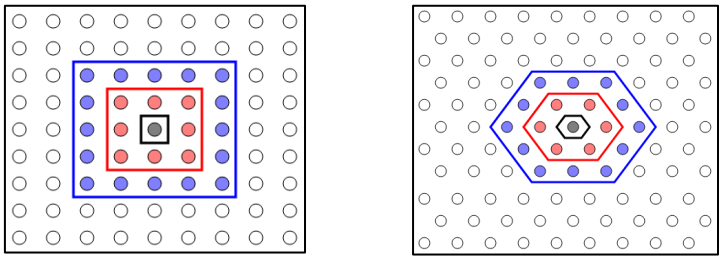
La carte de Kohonen, aussi appelée carte auto adaptive (Self Organizing Map - SOM) a été créée en 1984 par Teuvo Kohonen et se classe parmi les modes d'apprentissage non supervisé. En effet, le but est d'organiser des données discrétisées et représentées par des vecteurs associés à des neurones faisant partis d'une carte (grille). A chaque neurone est ensuite associé un poids qui sera amené à évoluer pendant la phase d'apprentissage.

Le principe est de parcourir l'ensemble des données et de les comparer avec les différents neurones. Celui qui "réagira" (correspondra) le mieux, sera alors le neurone dit gagnant et représentera les données. L'ensemble des neurones entourant le neurone gagnant seront également affectés afin qu'ils se rapprochent d'avantage des données de ce dernier. La phase d'apprentissage se termine lorsque tous les neurones sont stables.

Les cartes associées peuvent être de dimensions différentes :

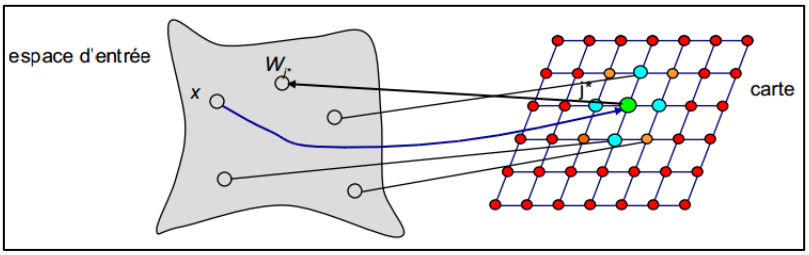


Unidimensionnelle

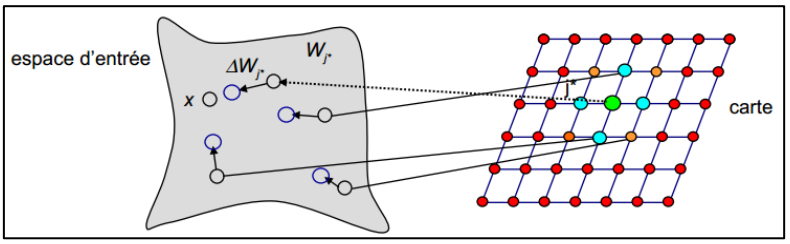


Bidimensionnelle

Exemple d'organisation sur une grille bidimensionnelle de type carrée, c'est à dire avec 4 voisins.



Sélection du neurone le plus proche de l'exemple X



Mise à jour des poids des neurones entourant le neurone gagnant

Comme expliqué précédemment, chaque neurone de la carte est associé à un poids noté *W*. Lors de la phase d'apprentissage, les itérations sont notées *t*.

A chaque itération *t*, un exemple d'apprentissage *X(t)* est choisi et comparé à l'ensemble des neurones. Le neurone gagnant *j** est celui dont le vecteur poids $W_{j^*}(t)$ est le plus proche de *X(t)*.

$$d_N [X(t) , W_{j^*}(t)] = min_j \, d_N [X(t) , W(t)]$$

avec :

- d_N distance dans l'espace d'entrée

La détermination des neurones voisins du neurone gagnant se fait de la manière suivante:

$$h_{j^*} (j , t) = h [d(j , j^*), t]$$

avec :

- *d* distance dans la carte
- *h* fonction de voisinage

Puis, la mise à jour des poids se fait ainsi :

$$W_j (t + 1) = W_j (t) + \Delta W_j (t)$$
$$\Delta W_j (t) = \varepsilon(t) . h_{j^*} (j , t) . (X(t) - W_j (t))$$

avec :

- ε pas d'apprentissage

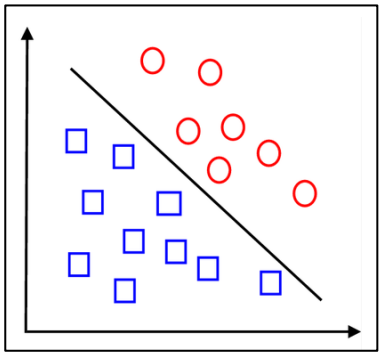
Classifieur linéaire

Un classifieur linéaire représente un type d'algorithmes d'intelligence artificielle qui va permettre de trier des échantillons à partir d'une combinaison linéaire des propriétés de chaque échantillon. Cette combinaison est aussi appelée fonction de décision (ici *g(x)*, mais on la trouve souvent noté *h(x)*). Elle s'exprime donc par une fonction linéaire suivant les propriétés de chaque échantillon (dans notre cas noté *x*). Chacune de ces propriétés est soumise à un poids qui évoluera au cours de l'apprentissage (appelé *w*), généralement on ajoute un biais comme

propriété à chaque échantillon ainsi qu'un poids qui sera lui aussi modifié au cours du temps (qui est noté *w0* ou encore *b*). Et enfin, la fonction *f()* est une fonction qui permet de convertir le produit scalaire entre deux propriétés en la sortie attendue, cette fonction peut être la fonction de signe, la fonction de Heaviside ou encore la fonction sigmoïde. Ci-dessous vous pouvez voir la formule qui résume ce que fait un classifieur linéaire.

$$g(x) = f(w^T x + w_0) = f\left(\sum_{j=1}^{N} w_j x_j + w_0\right)$$

Finalement, la représentation graphique de ce genre de classifieur linéaire revient à faire une séparation avec un hyperplan entre les données de deux classes différentes, comme on peut le voir dans la figure suivante.



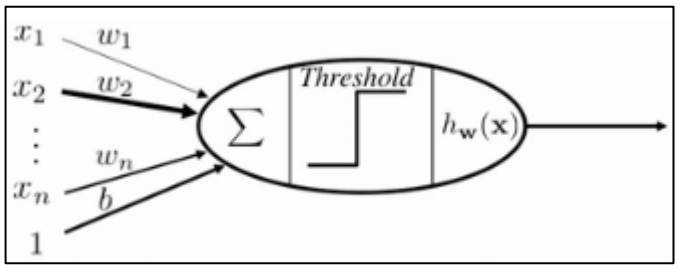
Cependant, les classifieurs sont basés sur les probabilités, ce qui peut parfois introduire des erreurs de classement, mais plus le nombre d'échantillons est important et meilleurs seront les résultats.

Perceptron

Le perceptron est un classifieur linéaire inventé en 1957 par Frank Rosenblatt qui a été inspiré par les théories cognitives de Friedrich Hayek et de Donald Hebb. Le perceptron est souvent considéré comme le plus simple des réseaux de neurones, et il est de type supervisé.

L'idée générale du perceptron est de modéliser une décision à partir d'une fonction linéaire et d'y ajouter une fonction de seuil (comme on l'a vu précédemment avec le classifieur linéaire). Nous avons donc en entrée un échantillon/vecteur que

l'on nomme x, qui est décomposé en plusieurs éléments qui sont notés de x_1 à x_n. Sur chacun de ces éléments sont appliqués des poids de w_1 jusqu'à w_n, ensuite il faut appliquer le produit scalaire de tous ces poids aux éléments du vecteur : $x_1.w_1,...,x_n.w_n$, et en faire la somme. Le résultat ainsi obtenu sera donné en entrée à la fonction de seuil (noté ici Threshold) qui va permettre de savoir si cet élément correspond à la classe 1 si le résultat de la somme est supérieur ou égal à 0 sinon il appartient à la classe 0.



On ajoute également le biais comme on l'a vu pour le classifieur linéaire, afin de faire varier le seuil de comparaison et donc d'avoir une bonne classification des données.

Le perceptron a tout de même ses limites, il ne peut résoudre que des problèmes linéaires séparables. Il ne peut par exemple pas résoudre le cas du XOR ("ou-exclusif") à lui seul, mais une utilisation d'un perceptron multi-couches permettra de résoudre ce problème.

Algorithme Perceptron simple couche

L'algorithme du perceptron simple couche est assez facile à mettre en place. Pour cela, il faut dans un premier temps définir la valeur du taux d'apprentissage et ensuite suivre les consignes suivantes :

Tant qu'on a pas obtenu le critère d'arrêt défini (soit le nombre d'erreurs vaut 0, soit le nombre maximal d'itérations est atteint)
    Pour chaque vecteur de la base d'entraînement x_t
        Calcule du produit vectoriel de chaque élément du vecteur par son poids
        Récupération de la valeur de sortie de Threshold(x_t . w) dans $h(x_t)$
        Si $h(x_t)$ est différent du résultat attendu y_t
            Pour tous les éléments du vecteur et le biais ($x_{t,i}$)
                Mettre à jour des poids avec la formule suivante
                $w_i \leftarrow w_i + TAUX\_APPRENTISSAGE * (y_t - h(x_t)) * x_{t,i}$
            Fin Pour
        Fin Si
    Fin Pour
Fin Tant que

Dans ce pseudo-code, on retrouve bien la formule mathématique qui a été expliquée précédemment. Vous pourrez retrouver directement cette implémentation d'algorithme en C++ dans l'annexe sur un exemple quelconque.

Rétropropagation du gradient de l'erreur

Cet algorithme a été inventé en 1984 par Paul J. Werbos et a été mis en place pour la première fois en 1986 par David Rumelhart. Le but de cet algorithme est de minimiser les erreurs dues à la différence entre la/les sorties attendues et celle(s) obtenue(s), en recalculant les poids des synapses de chaque neurone. Pour cela, on effectue une modification de ces poids en partant de la dernière couche vers la première.

Afin de réaliser la rétropropagation du gradient, la méthode à appliquer est celle mettant en place les dérivées partielles, ainsi que ce que l'on appelle le gradient.

Fonction de minimisation de l'erreur

On cherche à minimiser l'erreur entre la sortie attendue (y_t) et celle obtenue ($h_w(x_t)$). Pour la fonction de seuil (Threshold) que l'on a vu précédemment, on va donc effectuer la différence entre ces deux valeurs que l'on multiplie ensuite par le produit scalaire du poids de l'élément (w) par la valeur de cet élément (x_t).

$$Loss(y_t, h_{\mathbf{w}}(\mathbf{x}_t)) = -(y_t - h_{\mathbf{w}}(\mathbf{x}_t))\mathbf{w} \cdot \mathbf{x}_t$$

La fonction de minimisation est différente suivant le type de fonction d'activation du neurone. Si nous prenons la fonction sigmoïde qui permet de faire des probabilités sur l'appartenance d'un élément à une classe, nous allons obtenir une fonction de minimisation plus complexe que celle de la fonction de seuil.

$$h_{\mathbf{w}}(\mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w} \cdot \mathbf{x}}}$$

Fonction de décision de la fonction sigmoïde

$$Loss(y_t, h_{\mathbf{w}}(\mathbf{x}_t)) = -y_t \log h_{\mathbf{w}}(\mathbf{x}_t) - (1 - y_t) \log(1 - h_{\mathbf{w}}(\mathbf{x}_t))$$

Le résultat ainsi obtenu par cette fonction de seuil, nous permet de savoir si la prédiction a été correctement faite avec 0, dans le cas contraire la prédiction est mauvaise, et par conséquent, on va pouvoir calculer le seuil à dépasser pour qu'elle soit bonne.

Les dérivées partielles

Une dérivée partielle correspond à l'état plus général d'une dérivée. Elle s'applique aux fonctions qui dépendent de plus d'un élément. Pour effectuer cette dérivée partielle, on va dériver notre fonction sur chacune des variables en considérant les autres variables comme des constantes.

$$\frac{\partial f(x,y)}{\partial x} = \lim_{\Delta \to 0} \frac{f(x+\Delta, y) - f(x,y)}{\Delta}$$

$$\frac{\partial f(x,y)}{\partial y} = \lim_{\Delta \to 0} \frac{f(x, y+\Delta) - f(x,y)}{\Delta}$$

Dans le cas ci-dessus, nous avons une fonction *f* qui dépend de deux variables. Nous allons donc faire la dérivation dans un premier temps sur *x*, puis dans un second temps sur *y*.

Dans le cas de la fonction de seuil, les dérivées partielles sont:

$$p(y_t = 1|\mathbf{x}) = 1 - h_{\mathbf{w}}(\mathbf{x}_t)$$
$$p(y_t = 0|\mathbf{x}) = 0 - h_{\mathbf{w}}(\mathbf{x}_t)$$

Tandis que dans le cas de la fonction sigmoïde, les dérivées partielles sont :

$$p(y_t = 1|\mathbf{x}) = h_{\mathbf{w}}(\mathbf{x}_t)$$
$$p(y_t = 0|\mathbf{x}) = 1 - h_{\mathbf{w}}(\mathbf{x}_t)$$

Le gradient

Il correspond à un vecteur contenant toutes les dérivées partielles d'une fonction f. Pour illustrer cela, on reprend notre exemple précédent, et on note le gradient ainsi obtenu.

$$\nabla f(x,y) = \left[\frac{\partial f(x,y)}{\partial x}, \frac{\partial f(x,y)}{\partial y}\right]$$

Rétropropagation

Il faut maintenant calculer le gradient de ces pertes afin d'effectuer la rétropropagation. Pour cela, on peut utiliser plusieurs méthodes. La première méthode correspond à la forme basique de la rétropropagation qui nécessite de calculer la moyenne des dérivées partielles pour tous les vecteurs de la base de connaissance avant de pouvoir faire la mise à jour des poids des synapses, donc énormément de calcul avant de faire cette mise à jour.

La deuxième méthode, que l'on appelle la descente de gradient stochastique, permet de mettre à jour les poids en fonction du gradient d'une seule entrée de la base de connaissance prise au hasard (les poids étant bien entendu initialisés au départ de l'algorithme), on répète cette opération pendant un certain nombre d'itérations que l'on définit arbitrairement ou lorsque le seuil d'erreur est atteint. Lors de la mise à jour des poids, on applique également un facteur d'apprentissage (α) qui correspond à notre avancée pour obtenir l'erreur la plus petite possible. L'avantage de cette méthode est d'autant plus efficace dès lors que notre base de connaissance est importante, car on peut justement arrêter l'algorithme plus rapidement et faire beaucoup plus de mises à jour des poids.

$$w_i \leftarrow w_i - \alpha \frac{\partial}{\partial w_i} Loss(y_t, h_{\mathbf{w}}(\mathbf{x}_t)) \quad \forall i$$

Un troisième algorithme existe qui est la descente de gradient stochastique par mini-lots. Cette méthode consiste à effectuer des moyennes de dérivées partielles d'un certain nombre d'éléments de la base de connaissance (10, 20, 100, un valeur que l'on trouve pertinente). L'avantage est d'effectuer non pas des produits d'un vecteur avec matrice (poids avec tous les éléments d'une entrée de la base), mais d'effectuer un produit matriciel entre les poids des éléments sélectionnés et la valeur de ces éléments. Cette solution est en réalité un mixte des deux précédents algorithmes, si on augmente trop le nombre d'éléments à traiter, on arrive dans le traitement de la rétropropagation basique, tandis que si on rabaisse le nombre d'éléments à 1, l'algorithme devient de la rétropropagation stochastique.

Il y a encore un autre algorithme que l'on appelle Resilient Propagation (RProp) créé en 1992 par Martin Riedmiller et Heinrich Braun qui, lui, ne prend pas en compte de facteur d'apprentissage. Il se base seulement sur les signes des dérivées partielles de tous les éléments du vecteur d'entrée. En effet, si le signe de la dérivée est le même qu'à la précédente itération, alors on pondère le poids par le facteur η^+ qui est la plupart du temps mis à 1.2. Par contre, si le signe est différent c'est qu'on a dépassé le minimum local et donc on va changer le signe du poids de notre neurone et le multiplier par η^- qui est généralement positionné à 0.5. Enfin, le dernier cas c'est si les valeurs des dérivées n'ont pas changé alors on va continuer à

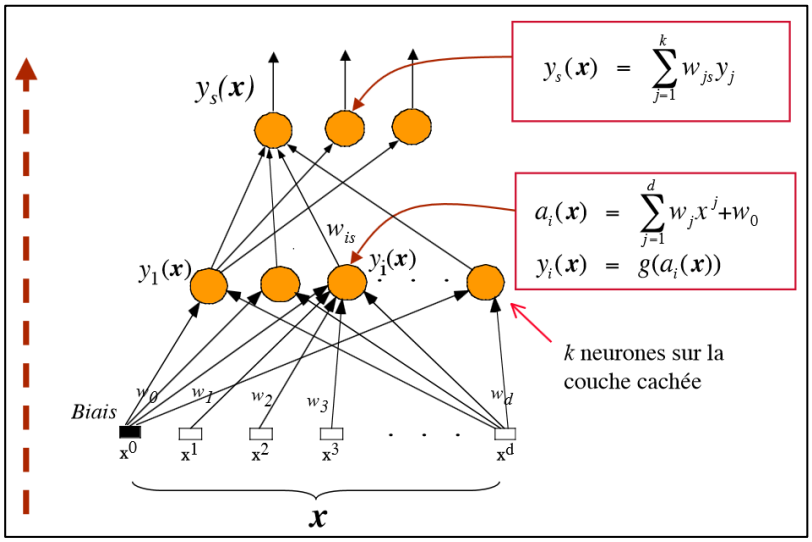
appliquer les changements faits à l'itération précédente. Avec cette dernière méthode, on effectue moins de calcul et donc on gagne en performance.

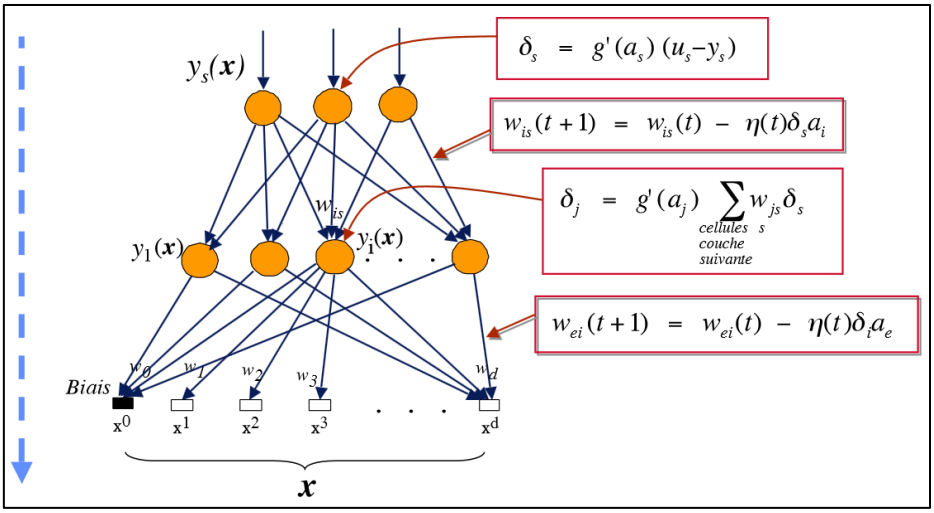
De plus, afin de sortir des minimums locaux, on ajoute un terme d'inertie (appelé aussi momentum) qui est pondéré avec un taux. Lors de chaque de boucle de mise à jour des poids, on conserve les informations de changement des poids. Cette inertie permet d'optimiser de le réseau de neurone et d'éviter les oscillations des poids.

Algorithme Perceptron multi-couches

Le perceptron multi-couches est un réseau de neurones artificiels où les informations vont toujours vers la sortie. La toute première couche du réseau de neurones correspond aux éléments qui composent notre vecteur d'entrée ; tandis que la couche finale correspond à la sortie du réseau. Tous les neurones sont reliés par des liaisons pondérées, comme on l'a vu précédemment avec le perceptron simple couche entre les entrées et la fonction de seuil.

La partie apprentissage du perceptron multi-couches est basée sur la rétropropagation du gradient de l'erreur, on cherche donc à faire une recherche de minimisation de l'erreur entre la sortie obtenue et celle attendue.





Sens de la rétropropagation de l'erreur

L'algorithme du perceptron est finalement assez simple à mettre en place, il peut être découpé en trois étapes :

1- La propagation des entrées jusqu'à la couche de sortie

2- Le calcul de l'erreur en sortie et rétropropagation de l'erreur

3- La correction des poids

Plusieurs paramètres entrent en compte dans le déroulement de cet algorithme, il y a, dans un premier temps, le type de la fonction d'activation (de décision) des neurones, par exemple, la fonction de seuil ou sigmoïde, et dans un second temps, le nombre d'entrées de la base de connaissance qui va influencer le choix de la méthode de rétropropagation du gradient de l'erreur.

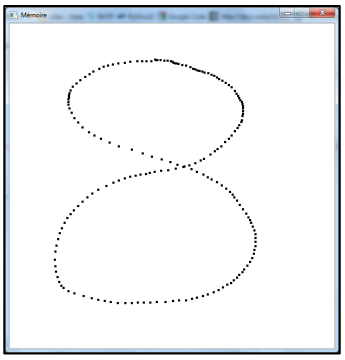
Un exemple d'un perceptron multi-couches est présent en annexe, dans cet exemple, nous avons intégré la possibilité de choisir entre la fonction de seuil et sigmoïde comme choix d'activation, et nous avons mis en place la rétropropagation du gradient stochastique. Son but est d'apprendre à la fonction XOR.

Le perceptron multi-couches est présent dans les jeux vidéo tels que Colin McRae 2.0 afin d'améliorer les IA du jeu et donc pour obtenir la meilleure conduite possible. Dans leur algorithme, les développeurs ont utilisé la méthode de rétropropagation du gradient RPROP car ils le considèrent comme le plus rapide. Ils ont décidé de mettre en place ce genre d'algorithme car le nombre de cas à prévoir pour la gestion de la conduite était trop important et trop complexe.

Projet

Description du projet

Le projet, que nous allons développer afin d'illustrer une méthode de classification, consiste à réaliser de la reconnaissance de formes, ces dernières étant réalisées par un utilisateur. L'apprentissage des formes sera effectué par un perceptron multicouches. Les formes, que l'utilisateur dessinera, seront des représentations de comportement d'ennemis, c'est-à-dire que l'utilisateur va dessiner la forme ci-dessous, et notre application pourra lui indiquer quel comportement il a représenté, c'est-à-dire un comportement de ronde en forme de 8. La partie de récupération des données est faite par un logiciel que nous avons développé qui a été fait en C++ avec OpenGL 2.0.

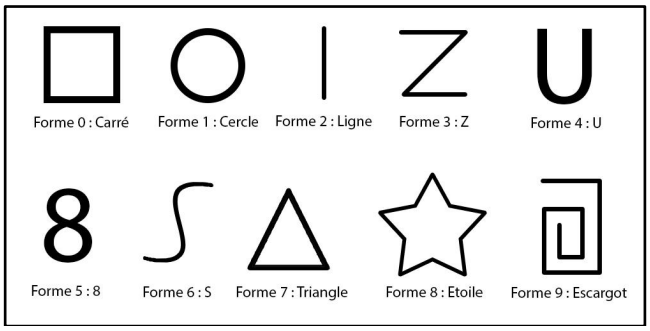


Les données, que nous récupérons, sont les angles formés entre les différents points de la forme, nous avons choisi les angles comme valeurs de référence car ils permettent de gérer les différentes tailles et rotations. Nous ne gardons pas tous les angles formés car le perceptron doit utiliser un nombre constant de valeurs d'entrée, donc d'angles. Nous allons donc effectuer des tests en prenant également en compte le nombre d'angles fournis. Les données sont sauvegardées dans un fichier texte, de la façon suivante: les différents d'angles séparé par un espace et à la fin de la ligne sera présent l'indice de la forme voulue. Ci-dessous se trouve une représentation des données formant la base de connaissance. Les angles ont un format bien particulier, ils sont à la base compris entre [0,360], puis nous les changeons d'échelle pour qu'ils soient compris entre [0,1].

0.286701 0.438347 0.612459 0.731328 0.68557 0.704773 0.564893 0.379599 0.394792 0.354375 0 0 0 0 0 1 0 0 0 0

Angles — Indice forme

Nous avons entraîner le programme à reconnaître les différentes formes ci-dessous, qui sont toutes des patterns que l'on pourrai retrouver dans un jeu de stratégie en temps réel (RTS). Nous étudierons pour chacun de ces patterns la variance des variables composant le programme afin de déterminer quelles sont les paramétrages les plus adaptés selon les patterns et pourquoi.



Pour l'algorithme du perceptron multi-couches, nous utilisons une librairie externe faite par Sylvain Barthélémy en C++, avec son algorithme seul la fonction sigmoïde est prise en compte. Nous avons légèrement modifié sa librairie afin de pouvoir tester une forme lorsque l'algorithme du perceptron a terminé son exécution. L'utilisation de son perceptron multicouches est rapide à mettre en place. En effet, nous devons simplement lui indiquer les caractéristiques du perceptron que nous souhaitons et le fichier contenant la base de connaissance.

```
int layers[] = { 10, 25, 10 };
MultiLayerPerceptron mlp(3, layers);
mlp.Run("./../OpenGL/Memoire/Memoire/donnees_10essais_10angles.txt", 5000);
mlp.Evaluate("./../OpenGL/Memoire/Memoire/donnees_8_10angles.txt");
```

Dans l'exemple ci-dessus, on a défini un perceptron acceptant 10 entrées, c'est-à-dire 10 angles, et 10 sorties, correspondant aux 10 formes que nous avons définies afin obtenir un pourcentage de correspondance pour chaque forme. On remarquera la présence d'une couche cachée, contenant 25 neurones. La fonction *Run* du perceptron permet de lancer l'apprentissage qui sera fait à partir de la base de connaissance de 10 essais par forme, et où chaque forme est définie par 10 angles. Le nombre 5000 indiqué à côté du nom du fichier est le nombre de boucles maximum exécutées par le perceptron, nous l'avons placé assez haut afin de permettre un grand nombre d'entraînements. Cependant, nous ne l'avons pas jugé important dans notre cas pour le faire varier car nous avons des données assez

différentes donc si le perceptron se termine en arrivant au bout des 5000 boucles, c'est qu'il souhaite encore peaufiner son apprentissage ou que les données de la base de connaissance sont trop disparates. Pour terminer, la fonction *Evaluate* permet d'obtenir les pourcentages d'appartenance de la forme donnée en entrée par rapport au perceptron.

Il reste deux paramètres que nous n'avons pas abordés qui sont le taux d'apprentissage et le taux d'inertie. Nous les avons tous les trois fixés afin de se concentrer sur l'étude des variances du nombre de neurones par couche cachée et du nombre de couche cachée. Le taux d'apprentissage est fixé arbitrairement à 0.25, tout comme le taux d'inertie à 0.9.

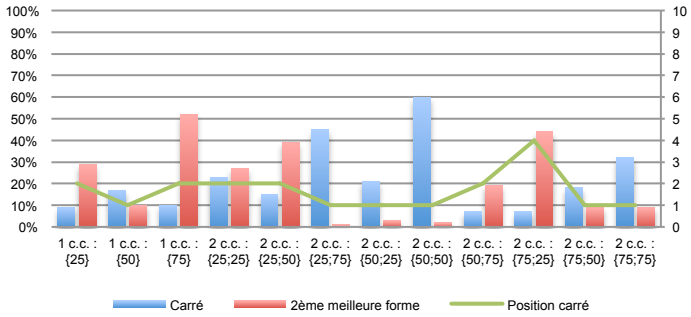
Etude des formes

Cette section est dédiée à l'étude des données établies sur l'ensemble des formes précédemment indiquées. 6 graphiques ont été créés par forme, chacun ayant pour base un nombre différent d'essais (10 et 20) pour un nombre d'angles de 10, 20 et 50. Pour chacune de ces configurations, le taux de reconnaissance (en pourcentage) de la forme est indiqué par le nom de la forme recherchée elle-même (représenté en bleu sur l'axe gauche). Plus ce taux est haut, plus cela signifie que la forme a été reconnue. Nous étudions également le pourcentage de reconnaissance de la forme ayant obtenu le meilleur taux après la forme recherchée (représenté en rouge sur l'axe gauche). Cela révèle que le programme a bien identifié la forme recherchée ou s'il "hésite" avec une autre. Dans le cas où la forme recherchée n'a pas obtenu le meilleur taux, alors la courbe rouge représente le taux le plus haut renvoyé par le programme. De plus, nous indiquons en quelle position la forme recherchée est arrivée sur la base des 10 formes exploitées (représenté en vert sur l'axe droit).

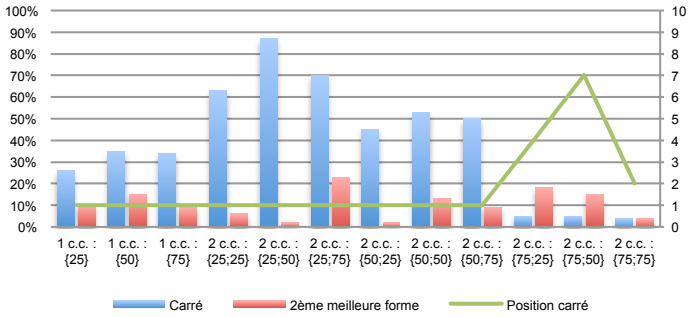
L'ensemble des tests a été fait avec différents paramétrages qui serviront à mettre en avant ce qui fonctionne le mieux pour chaque configuration. Les tests évolueront selon le nombre de couches cachées (indiqué par c.c. allant de 1 à 2) et le nombre de neurones les composant (ex: {25} pour 25 neurones dans la couche cachée simple, ou {25;75} pour 25 neurones sur la première couche cachée, et 75 sur la deuxième).

Etude de la forme "Carré"

10 Essais - 10 angles



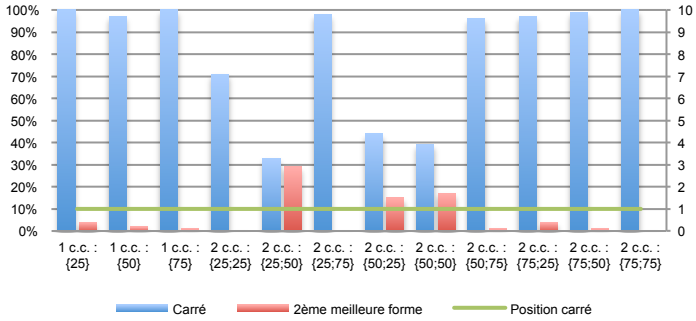
20 Essais - 10 angles



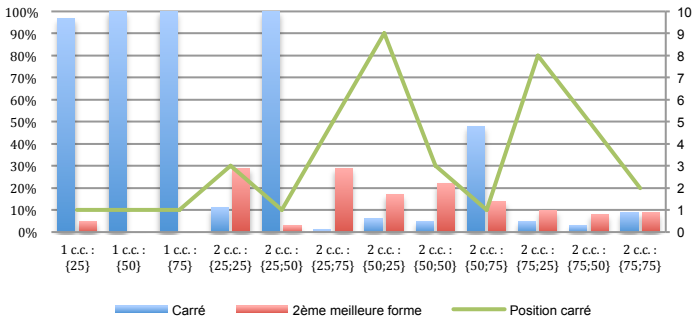
Commentaire

On constate que le programme s'est moins trompé en appliquant 20 essais à chaque test plutôt que 10. Cependant c'est aussi avec ce nombre d'essais que la forme a été la plus mal classée. Malgré tout, les tests positifs de ce graphique sont également ceux ayant les plus grands écarts de pourcentage, donc les résultats les plus probants. Le meilleur résultat de ces graphiques, représentant la reconnaissance de forme en utilisant 10 angles, est donc 20 essais avec 2 couches cachées, avec 25 neurones sur la première couche et 75 sur la deuxième.

10 Essais - 20 angles



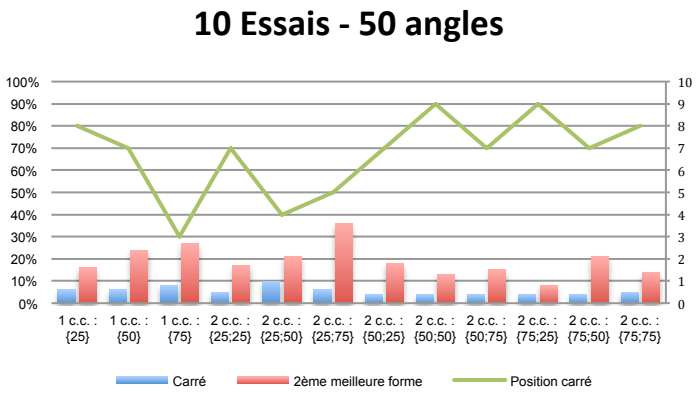
20 Essais - 20 angles

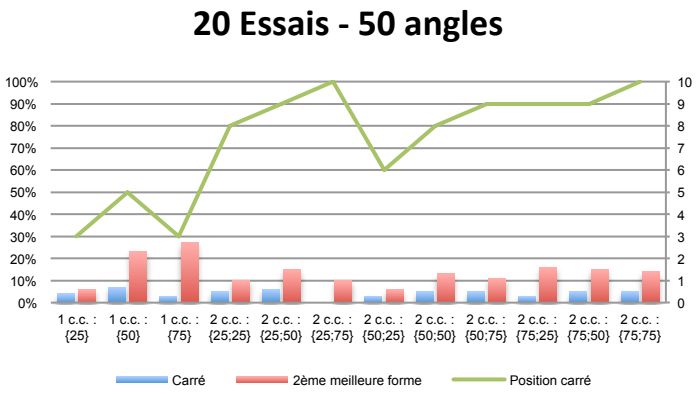


Commentaire

Le programme a fait moins d'erreur en appliquant seulement 10 essais à chaque test plutôt que 20. Sur le premier graphique, on constate que la forme recherchée a toujours été trouvée, et que dans 9 cas sur 12, presque qu'aucune autre forme n'a été trouvée. Cependant, sur le deuxième graphique, seulement

5 cas se sont révélé être positifs, cependant pour eux aussi presque qu'aucune autre forme n'a été détectée. Pour la majorité des cas négatifs, la forme recherchée a eu une très mauvaise position. Ces graphiques mettent en avant que 10 essais sont suffisants pour reconnaître la forme lorsqu'on teste sur 20 angles.





Commentaire

Les 2 graphiques montrent que l'expérience est un échec complet lorsqu'on utilise 50 angles. Aucun test n'a réussi à reconnaître la forme, et l'a en plus très mal placée en terme de positionnement. De plus on constate que l'ensemble des formes reconnues, correctes ou non, ont un très faible pourcentage de certitude, ce qui met en avant que le programme a pour ainsi dire rien reconnu.

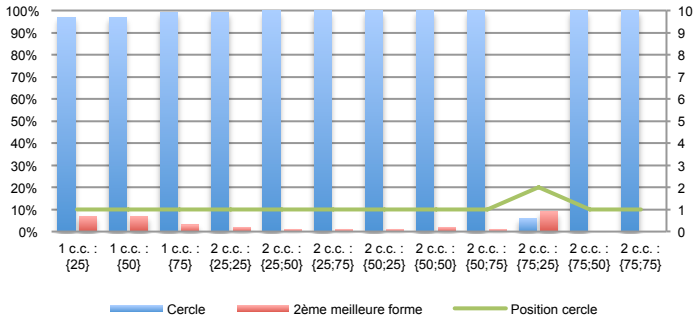
Conclusion

Sur l'ensemble des données exploitées, les taux de réussite sont plus présents dans les cas de "20 essais - 10 angles" et "10 essais - 20 angles". Malgré tout c'est le graphique "10 essais - 20 angles" qui se démarque par l'ensemble de ses résultats proches de 100%. On constate que l'ensemble des tests effectués sur 1 couche cachée sont tous probants, de même que ceux à 2 couches cachées ayant 75 neurones sur la première couche. Cependant, les données commencent à être erronées à partir de "20 essais - 20 angles" avec 2 couches cachées. On remarque qu'en plus d'avoir des données fausses, la position de la forme cherchée est très mauvaise. De même pour les 2 graphiques représentant 50 angles.

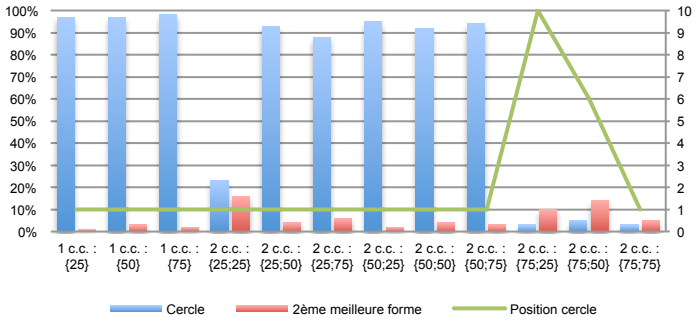
Ainsi le programme semble être plus a même de reconnaître la forme lorsqu'on applique 10 essais sur 20 angles.

Etude de la forme "Cercle"

10 Essais - 10 angles



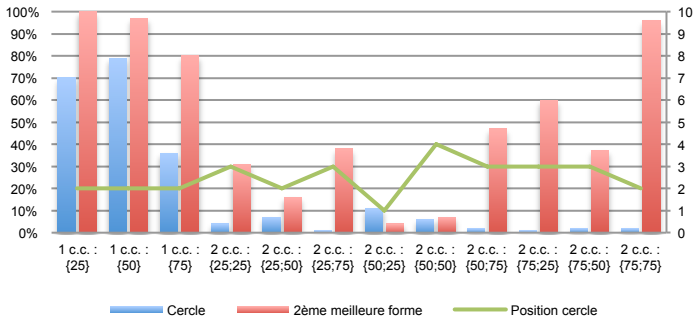
20 Essais - 10 angles



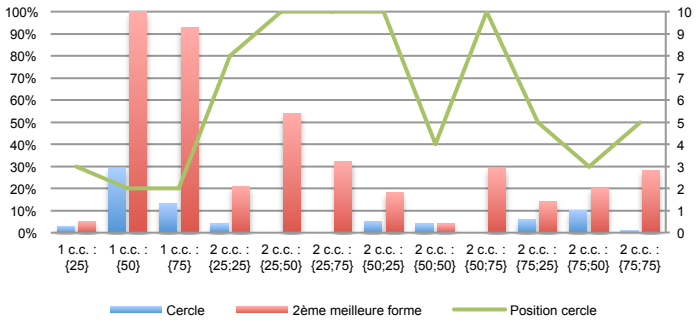
Commentaire

On constate que le programme s'est moins trompé en appliquant seulement 10 essais à chaque test plutôt que 20. Le cercle a été bien reconnu 11 cas sur 12 avec les 10 essais à plus de 95%, alors qu'avec les 20 essais, il a été bien reconnu à plus de 85% dans seulement 8 cas. Dans les cas des 2 couches cachées avec la première couche contenant 75 neurones avec 20 essais, la forme n'a pas été trouvée. Le meilleur résultat des ces graphiques représentant la reconnaissance de forme en utilisant 10 angles, est donc 10 essais avec 2 couches cachées, avec 25 neurones sur la première couche et 75 sur la deuxième.

10 Essais - 20 angles



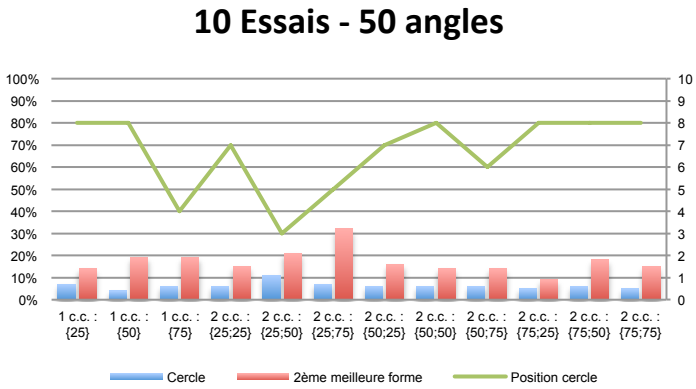
20 Essais - 20 angles

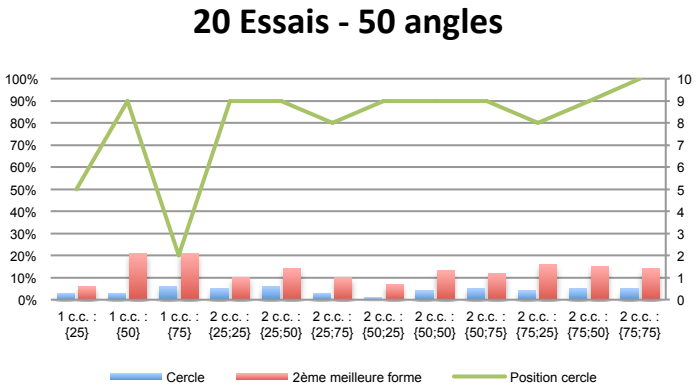


Commentaire

On remarque que la forme du cercle n'a jamais été trouvée aussi bien dans le cas des 10 essais que dans les 20, même si dans le cas des 10 essais avec les 2 couches cachées avec 50 neurones dans la première couche et 25 dans la

deuxième, la forme a été trouvée mais avec à peine plus de 10%. Cependant, dans le cas des 10 essais avec une couche cachée la forme a été trouvée en second, et parmi ces résultats, celui avec 50 neurones représente la meilleure solution.





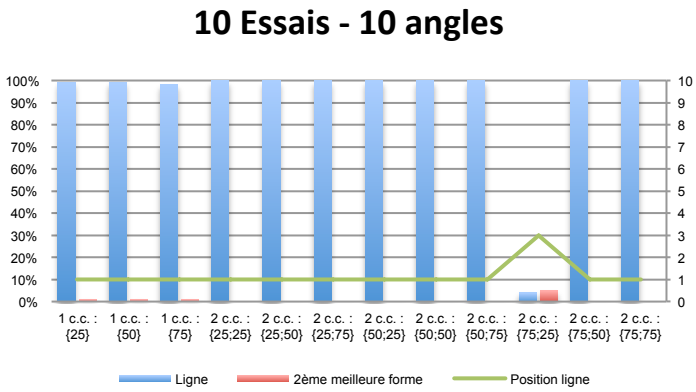
Commentaire

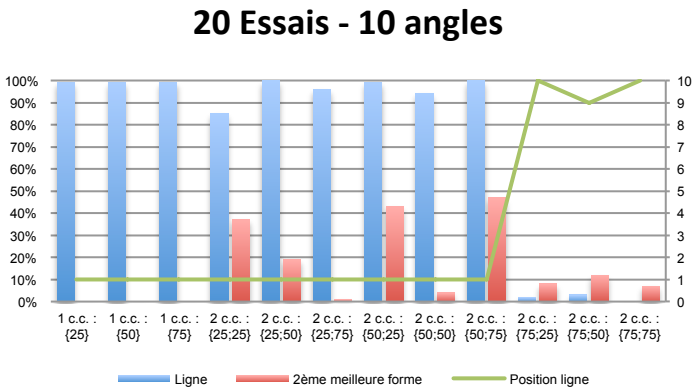
Dans ce cas, la forme n'a jamais été trouvée, et elle a toujours obtenu un résultat inférieure à 10%, sauf dans le cas des 10 essais avec 2 couches cachées de 25 et 50 neurones, où nous avons obtenu un résultat à peine supérieur à 10% et une troisième position, ce qui représente le meilleur résultat de cette série de tests. Cependant, nous ne pouvons pas dire que ces relevés soient de très bons résultats.

Conclusion

On constate que sur l'ensemble des données exploitées, les taux de réussite sont plus présents dans le cas de 10 essais avec 10 angles, mais le cas des 20 essais avec 10 angles obtient également des bons résultats dans la majorité des tests. On remarquera, cependant, que le cas des 2 couches cachées avec 50 neurones pour la première couche et 25 pour la deuxième a le meilleur rendement sur l'ensemble des tests effectués. A partir, des tests avec les 20 et 50 angles, les résultats sont totalement erronés et que la position de la forme devient de plus en plus mauvaise dans le classement.

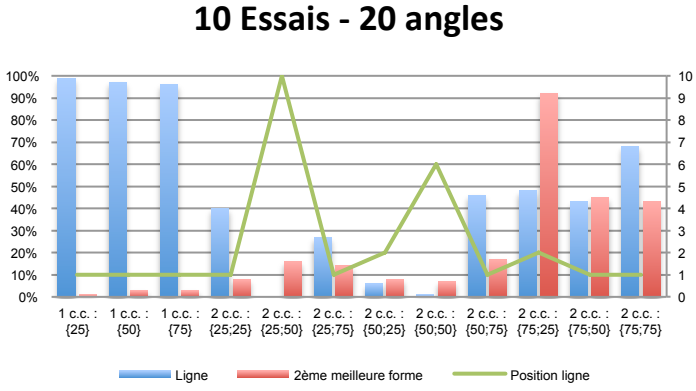
Etude de la forme "Ligne"

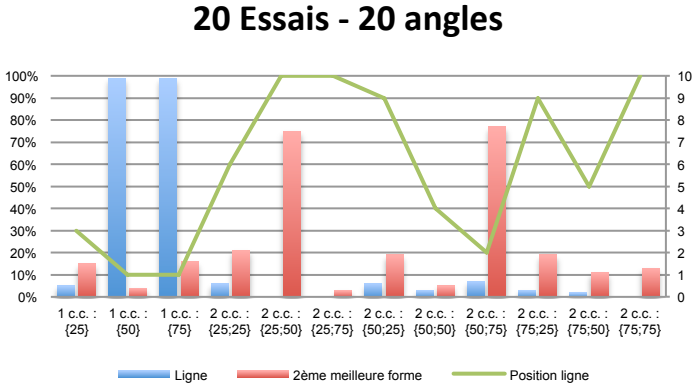




Commentaire

Le programme a dans l'ensemble bien reconnu la forme. En effet tous les tests concluants avoisinent les 100% de taux de reconnaissance, et presque tous n'ont pas de forme étrangère ayant été reconnue en plus de celle recherchée. Seulement 4 tests ont échoué sur les 24 en ne reconnaissant pratiquement aucune forme. De plus, 3 de ces tests ont placé la forme recherchée en très mauvaise position, ce qui indique que le programme ne partait pas du tout dans la bonne direction.

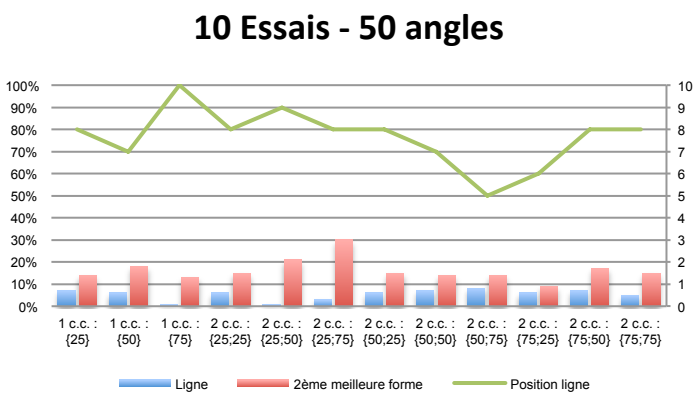


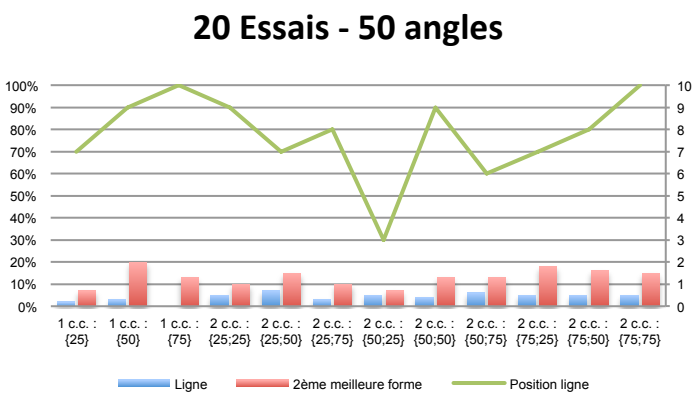


Commentaire

Les résultats de ces tests sont assez mitigés. 10 configurations ont reconnu la forme sur 24. Parmi celles-ci, 5 sont pratiquement à 100% de taux de reconnaissance, et le taux de leur forme concurrente est presque inexistant. Malgré tout, le programme s'est trompé 12 fois, et a très mal placé la forme recherchée. On

constate sur le deuxième graphique, qu'à partir de l'utilisation de 2 couches cachées, les tests sont négatifs mais ne trouvent presque qu'aucune forme sur les 10 proposées.





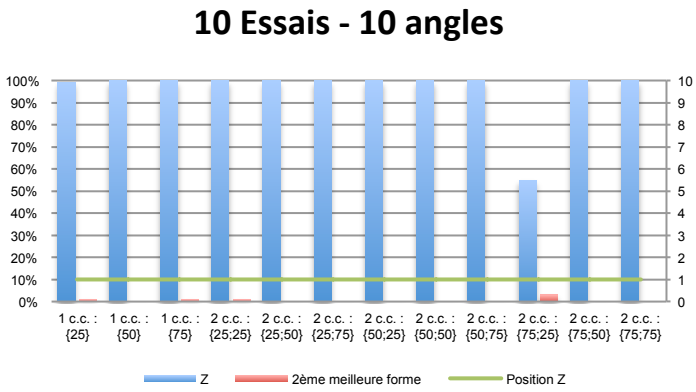
Commentaire

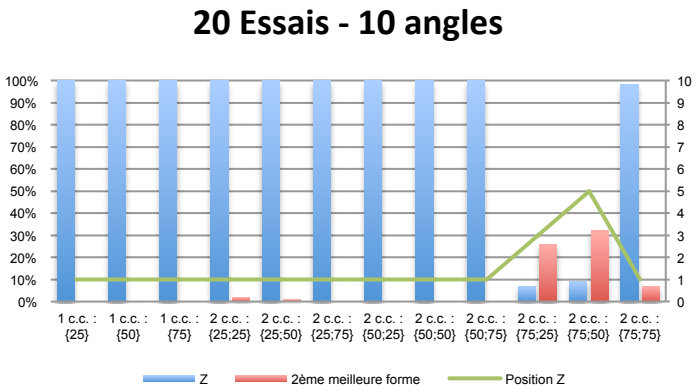
Les 2 graphiques montrent que l'expérience est un échec complet lorsqu'on utilise 50 angles. Aucun test n'a réussi à reconnaître la forme, et l'a en plus très mal placée en terme de positionnement. De plus, on constate que l'ensemble des formes reconnues, correctes ou non, ont un très faible pourcentage de certitude, ce qui met en avant que le programme a pour ainsi dire rien reconnu.

Conclusion

Sur l'ensemble des données exploitées, les taux de réussite sont plus présents dans les cas de "10 essais - 10 angles" et "20 essais - 10 angles". Malgré tout c'est le graphique "10 essais - 10 angles" qui se démarque par l'ensemble de ses résultats proches de 100%, même s'il comporte une erreur démontrant que le programme n'a presque rien reconnu. On remarque que les tests commencent à renvoyer de moins en moins de bons résultats dès qu'on passe à 20 angles. Le programme arrive à retrouver quelquefois la forme avec de très bon taux de reconnaissances, mais globalement c'est un échec. Les données empirent considérablement lorsqu'on passe 50 angles au programme, car en plus de ne presque rien reconnaître, il place très mal la forme recherchée. Ainsi le programme semble être plus à même de reconnaître la forme lorsqu'on applique 10 essais sur 10 angles.

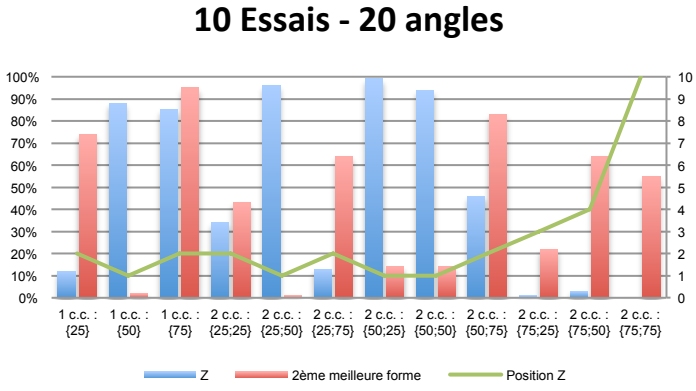
Etude de la forme "Z"

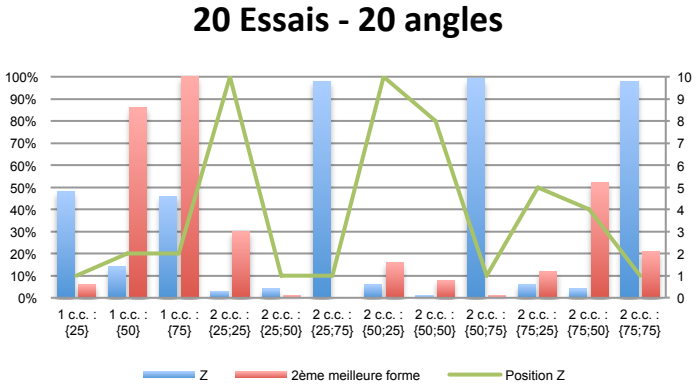




Commentaire

Sur ces graphiques, on remarque clairement que les tests avec 10 essais sont les plus probants. Cependant, on remarque que dans le cas des 2 couches cachées avec 75 neurones pour la première couche et 25 sur la deuxième, les résultats sont un peu moins probants dans le cas des 10 essais, et que clairement dans le cas des 20 essais, le résultat n'est pas bon, tout comme pour le cas des 2 couches cachées avec 75 neurones sur la première et 50 sur la deuxième. On peut parfaitement voir qu'ici plusieurs résultats sont "parfaits", c'est-à-dire que nous avons un taux de reconnaissance à 100% pour la forme recherchée et 0% pour les autres, donc ici il y a plusieurs bonnes solutions, comme celle d'une couche cachée avec 50 neurones.

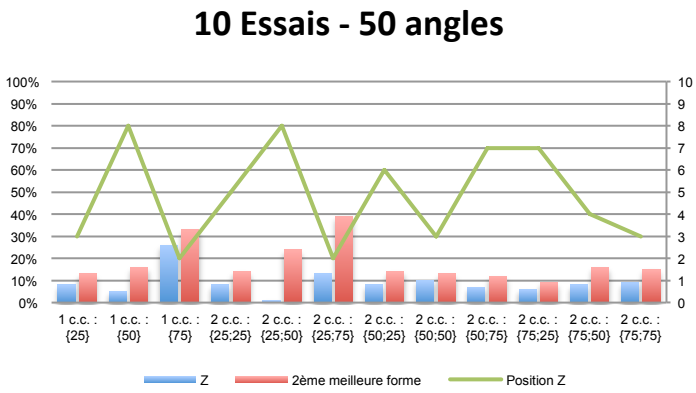


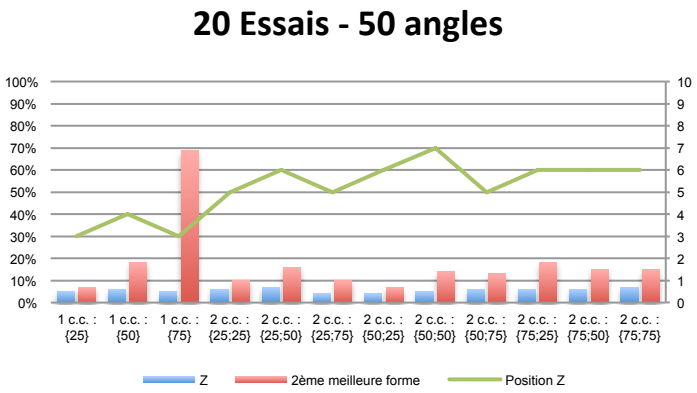


Commentaire

Dans ces tests, nous pouvons remarquer qu'un résultat ressort par rapport aux autres, le cas des 2 couches cachées avec 25 neurones sur la première couche

et 50 sur la deuxième. Les résultats sont très disparates suivant le nombre d'essais. Dans certains cas nous allons obtenir des résultats très probants avec 2 couches cachées avec 50 neurones pour la première et 25 sur la deuxième, alors que dans l'autre cas nous allons obtenir un résultat très mauvais.





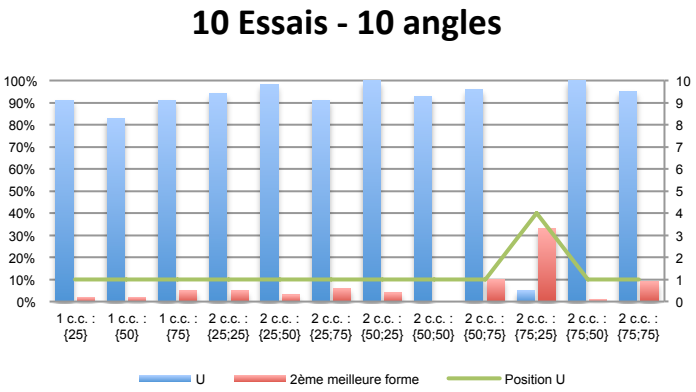
Commentaire

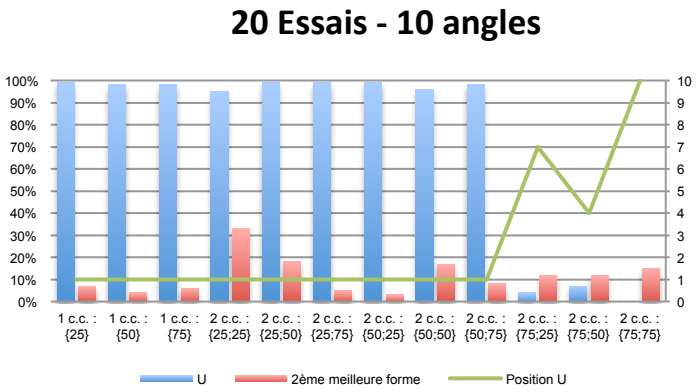
Dans ces tests, la forme n'a jamais été trouvée. Les pourcentages de correspondance entre la forme recherchée et celle trouvée sont généralement très proches, sauf dans 2 cas, et 1 tout particulièrement, à savoir celui de 1 couche cachée avec 50 neurones où l'autre forme obtient un résultat près de 70% alors que celle recherchée est d'environ 5% dans le cas des 20 essais. Cependant, on pourrait dire que le meilleur résultat obtenu est avec 10 essais et 1 couche cachée de 75 neurones, où la forme recherchée se place en deuxième position avec un score d'environ 25%.

Conclusion

Sur l'ensemble des données exploitées, les taux de réussite sont plus présents dans les cas de 10 essais avec 10 angles où la forme a toujours été trouvée, et où le taux d'erreur est inférieur à 5%. Le cas des 20 essais avec 10 angles obtient également de bons résultats malgré quelques erreurs. Cependant, dans le cas des 20 angles, les résultats deviennent plus mauvais, bien que la forme a été trouvé dans 4 cas sur les 12. Le cas des 50 angles est sans appel le plus mauvais de ces tests car la forme n'a jamais été trouvée.

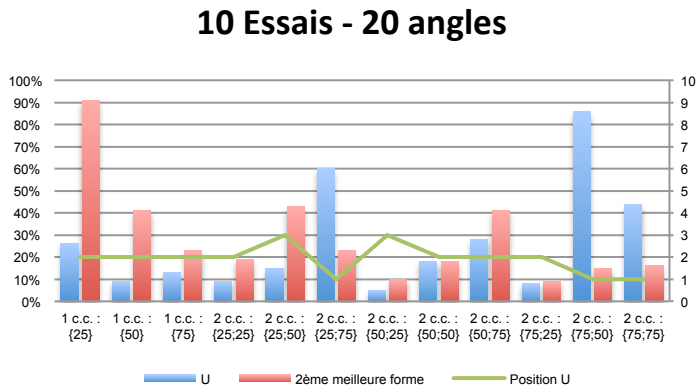
Etude de la forme "U"

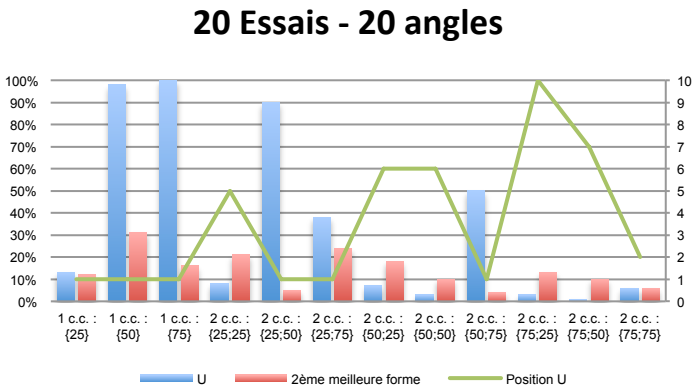




Commentaire

Le programme a dans l'ensemble bien reconnu la forme. En effet tous les tests concluant avoisinent les 100% de taux de reconnaissance, et presque tous n'ont pas de formes étrangères ayant été reconnues en plus de celles recherchées. Seulement 4 tests ont échoué sur les 24 en ne reconnaissant pratiquement aucune forme. De plus, ces tests ont placé la forme recherchée en très mauvaise position, ce qui indique que le programme ne se dirigeait absolument pas dans la bonne direction.

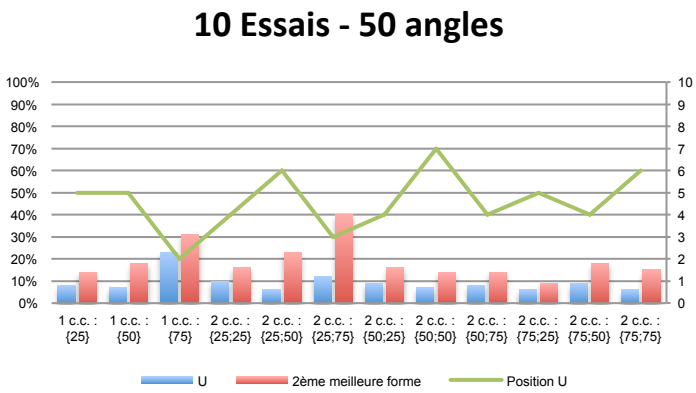


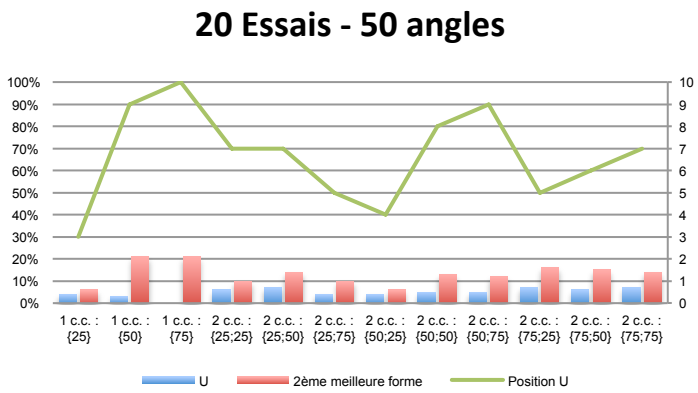


Commentaire

Les résultats de ces tests sont assez mauvais. 9 configurations ont reconnu la forme sur 24. Parmi celles-ci, 3 sont pratiquement à 100% de taux de reconnaissance, et 2 d'entre elles ont le taux, de la deuxième forme la plus proche,

presque inexistant. Le programme s'est trompé 13 fois, et a très mal placé la forme recherchée. Globalement, 20 angles ne semblent pas être un bon paramètre.





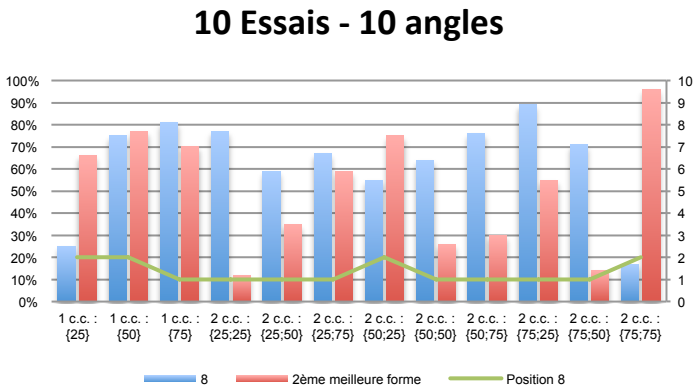
Commentaire

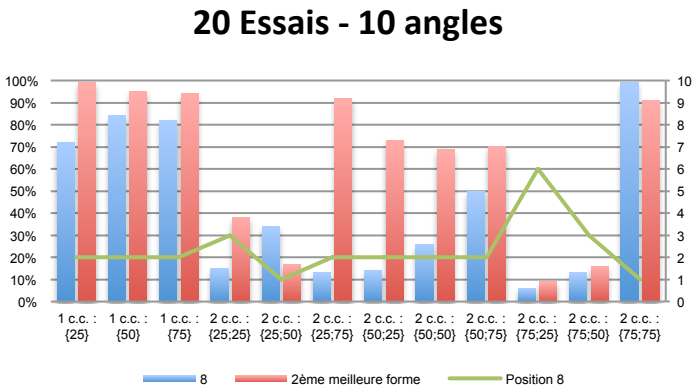
Les 2 graphiques montrent que l'expérience est un échec complet lorsqu'on utilise 50 angles. Aucun test n'a réussi à reconnaître la forme, et l'a en plus très mal placée en terme de positionnement. De plus on constate que l'ensemble des formes reconnues, correctes ou non, ont un très faible pourcentage de certitude, ce qui met en avant que le programme a pour ainsi dire rien reconnu.

Conclusion

Sur l'ensemble des données exploitées, les taux de réussite sont plus présents dans les cas de "10 essais - 10 angles" et "20 essais - 10 angles". Malgré tout c'est le graphique "10 essais - 10 angles" qui se démarque par l'ensemble de ses résultats proches de 100%, même s'il comporte une erreur démontrant que le programme n'a presque rien reconnu. On remarque que les tests ne renvoient plus de bons résultats dès qu'on passe à 20 angles. Le programme arrive à retrouver 2 fois la forme avec de très bon taux de reconnaissance, mais globalement c'est un échec. Les données empirent considérablement lorsqu'on passe 50 angles au programme, car en plus de ne rien reconnaître, il place très mal la forme recherchée. Ainsi le programme semble être plus a même de reconnaître la forme lorsqu'on applique 10 essais sur 10 angles.

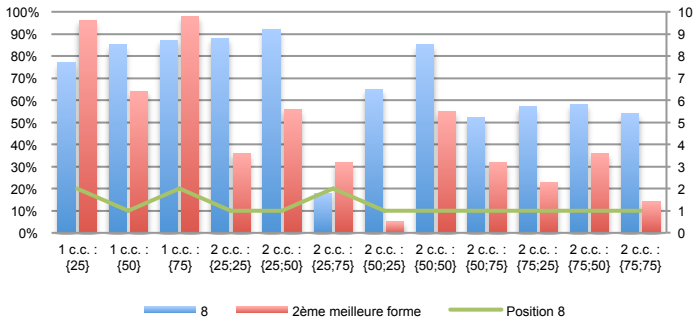
Etude de la forme "8"

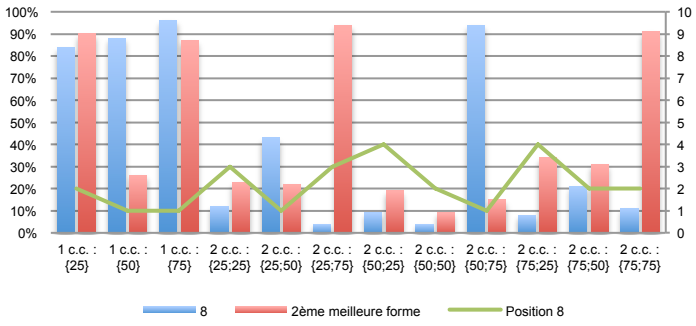




Commentaire

On constate que le programme s'est moins trompé en appliquant seulement 10 essais à chaque test plutôt que 20. De plus le programme fait apparaître de plus grands écarts de pourcentage entre la forme recherchée et la deuxième forme la plus proche, lorsqu'il utilisait 2 couches cachées. Le meilleur résultat de ces graphiques, représentant la reconnaissance de forme en utilisant 10 angles, est donc 10 essais avec 2 couches cachées dont la première est composée de 75 neurones et la deuxième de 25.

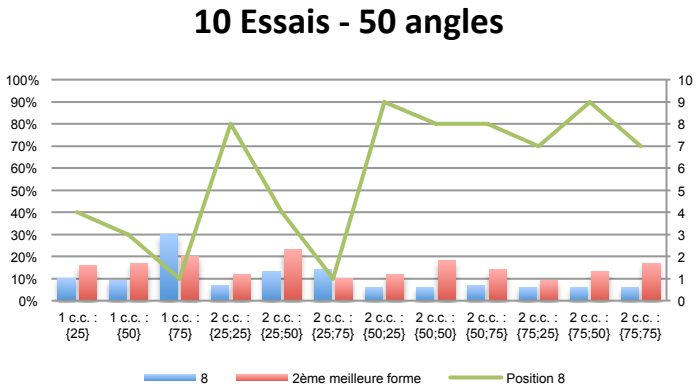


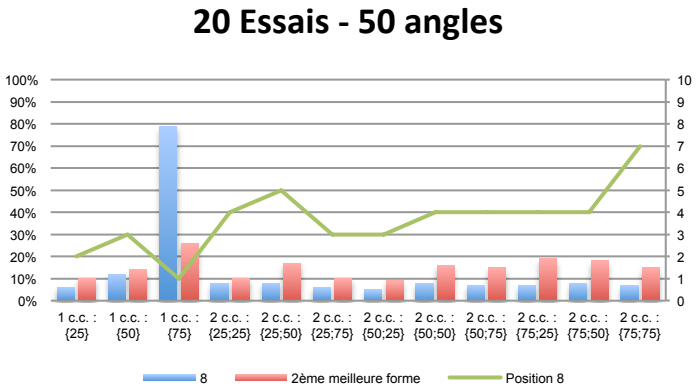


Commentaire

Le programme a fait moins d'erreur en appliquant 10 essais à chaque test plutôt que 20. De plus, en étudiant le deuxième graphique, on remarque que 5 cas sur les 8 négatifs ont des valeurs de pourcentage assez basses, ce qui signifie que le programme ne reconnaissait pas grand chose, et plus particulièrement pour le cas de la configuration : 2 couches cachées avec 50 neurones en première couche, et

50 en seconde, où le programme a reconnu au mieux une forme avec un pourcentage de 9%. Ces données mettent en avant qu'avec 20 essais et 20 angles, les configurations comprenant 2 couches cachées ne sont pas sûres pour cette forme. Le meilleur résultat de ces graphiques, représentant la reconnaissance de forme en utilisant 20 angles, est donc 20 essais avec 1 couche cachée et 75 neurones.





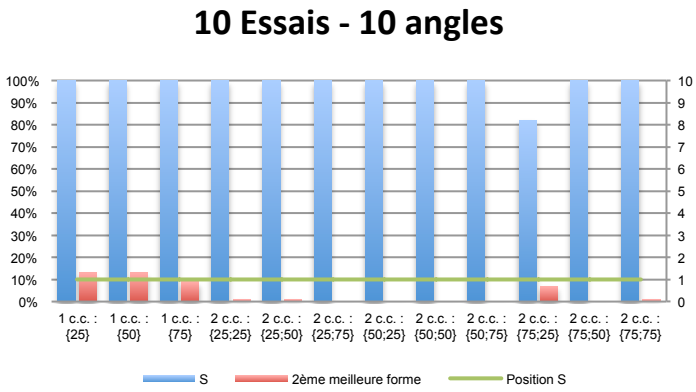
Commentaire

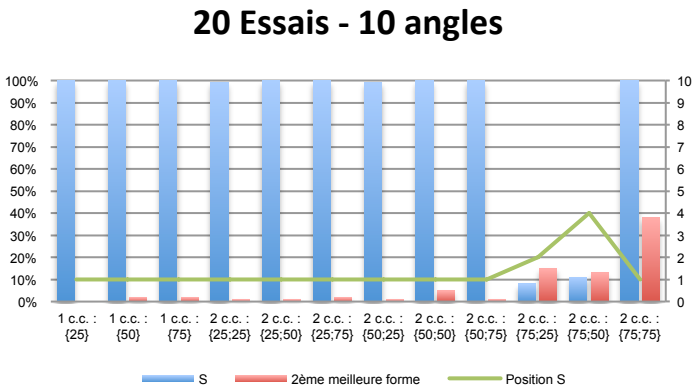
Les 2 graphiques montrent que l'expérience est globalement un échec lorsqu'on utilise 50 angles. Sur les 24 tests tous confondus, seulement 3 ont été positifs, dont 2 avec de très mauvais écart de pourcentage. De plus, les positions retournées de la forme sont très mauvaises. L'ensemble des données sur les 2 graphiques démontre que le programme ne reconnaît pas du tout la forme recherchée, mais aussi pratiquement aucune autre. Malgré tout, le meilleur résultat de ces graphiques représentant la reconnaissance de forme utilisant 50 angles, est donc 20 essais avec 1 couche cachée et 75 neurones.

Conclusion

On constate que sur l'ensemble des données exploitées, les taux de réussite sont plus présents dans les cas de 10 essais avec 10 et 20 angles. Les données les plus probantes de ces configurations étant celles à 2 couches cachées. Le meilleur cas est celui comprenant 10 essais avec 20 angles, 2 couches cachées dont la première avec 50 neurones et 25 sur la deuxième. Ce test est le meilleur car l'écart entre le pourcentage de la forme recherchée et celui de la forme la plus proche est de 65%, ce qui signifie que le programme est "sûr" de la forme retournée avec un écart de 65%.

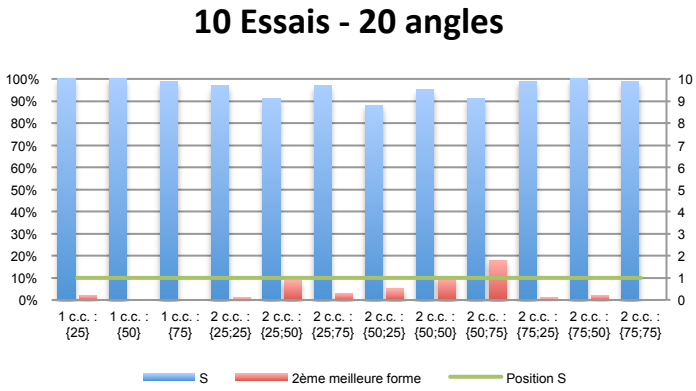
Etude de la forme "S"

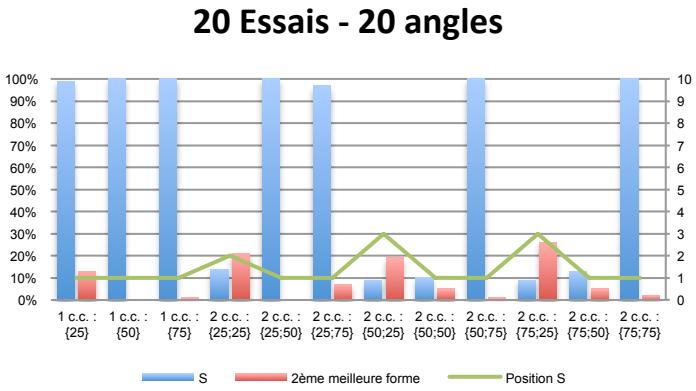




Commentaire

Le programme a dans l'ensemble bien reconnu la forme. En effet tous les tests concluants avoisinent les 100% de taux de reconnaissance, et presque tous n'ont pas de forme étrangère ayant été reconnue en plus de celle recherchée à l'exception d'un test sur le deuxième graphique. Seulement 2 tests ont échoué sur les 24 en ne reconnaissant pratiquement aucune forme. Globalement, ces tests sont une réussite.

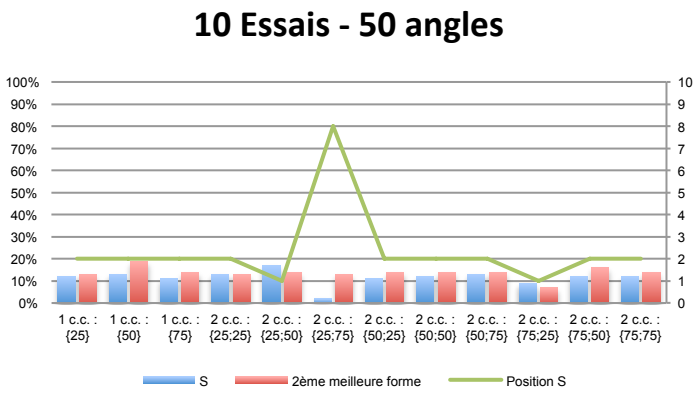


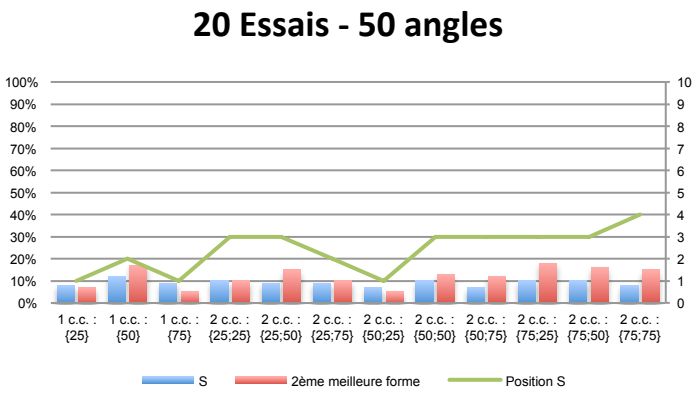


Commentaire

Les résultats du premier graphique sont très bons, avec des taux de reconnaissance presque à 100%, et des taux pour la deuxième forme presque nuls. Cependant les résultats du deuxième graphique sont mitigés. En effet 7 tests ont réussi avec presque 100% de taux de reconnaissance, mais les 5 autres n'ont presque rien reconnu. Ils n'ont par contre pas trop mal placé la forme recherchée.

Les tests effectués avec une couche cachée sont très bons, ce ne sont que ceux avec 2 couches cachées qui laissent à désirer.





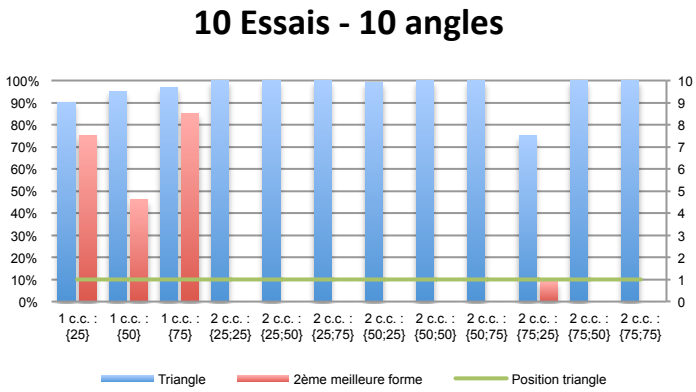
Commentaire

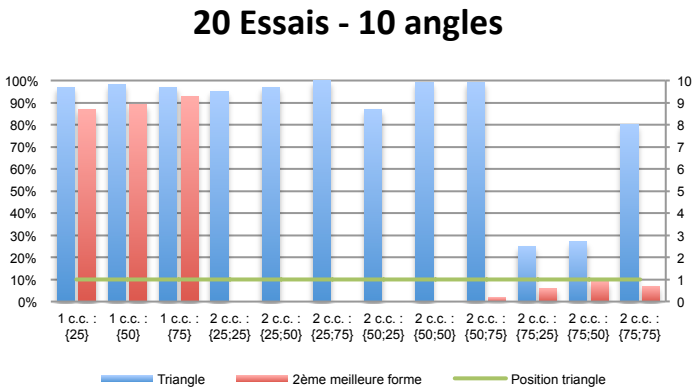
Les 2 graphiques sont globalement très mauvais. Seulement 5 tests ont réussi sur les 24, mais leur taux de reconnaissance est très faible ce qui signifie que le programme ne les a pas véritablement reconnus. Les résultats sont tellement mauvais, qu'on peut pratiquement les assimiler à des échecs. Hormis, le très faible taux de reconnaissance, la forme recherchée a globalement hérité d'une position pas trop mauvaise.

Conclusion

Dans cette étude 3 graphiques se démarquent par leurs très bons résultats. En effet des taux de reconnaissance de 100% ont pu être observés sur les graphiques "10 essais - 10 angles", "20 essais - 10 angles" et "10 essais - 20 angles". Malgré tout, c'est le graphique "10 essais - 10 angles" qui semble avoir les meilleures données, car il est celui qui a le moins de reconnaissance de deuxième forme. On constate que le graphique "20 essais - 20 angles" comporte moins de bonnes données, et plus particulièrement quand les tests avec 2 couches cachées commencent. Quant aux tests sur les 50 angles, même si le programme a parfois reconnu les formes, les taux de reconnaissance sont tellement faibles qu'on ne peut si référer. Au final, malgré que les positions renvoyées pour la forme recherchée ne soient pas trop mauvaises, l'ensemble des tests sur 50 angles sont des échecs. Ainsi le programme semble être plus à même de reconnaître la forme lorsqu'on applique 10 essais sur 10 angles.

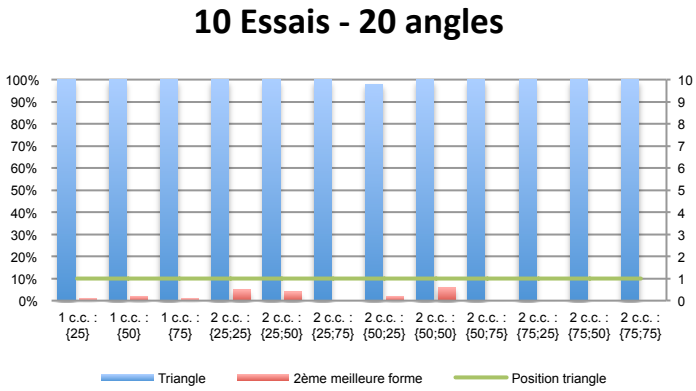
Etude de la forme "Triangle"

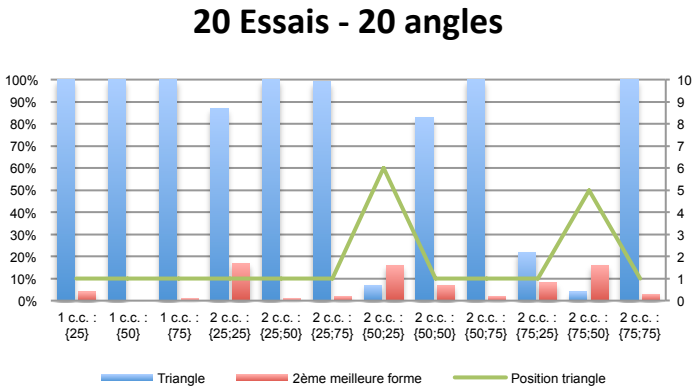




Commentaire

Dans les 2 cas, nous remarquons que la forme a toujours été trouvée, malgré tout nous constatons un fort taux de reconnaissance sur les tests avec 1 couche cachée. Cependant, nous pouvons remarquer que dans les cas des 20 essais avec 2 couches cachées de 75 neurones en première couche ainsi que 25 et 50 neurones en deuxième couche, ont des pourcentages de reconnaissance assez faibles d'environ 25%.

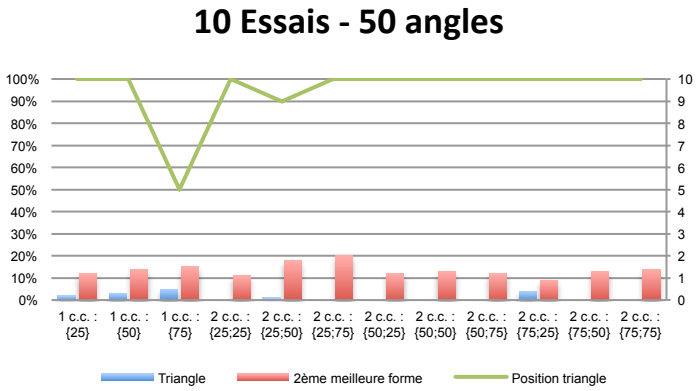


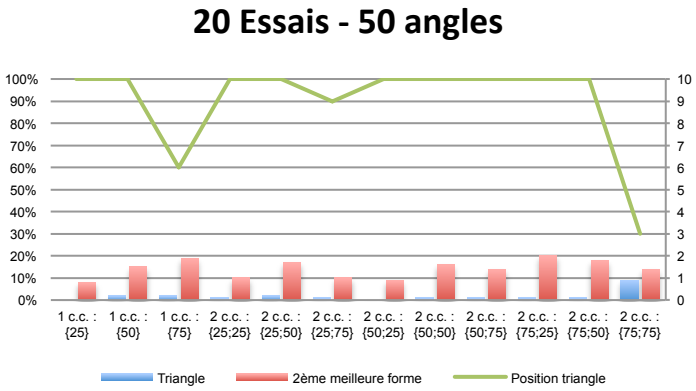


Commentaire

Les 2 graphiques ont de bons résultats, surtout celui des 10 essais qui est quasiment parfait avec un taux de reconnaissance près de 100% tandis que la reconnaissance des autres formes est sous la barre des 10%, alors que dans le cas des 20 essais, nous remarquons qu'il y a 3 résultats qui s'avèrent être moins bons.

Deux de ces résultats sont placés sur ceux des 2 couches cachées avec 75 neurones en première couche avec 25 et 50 neurones en deuxième couche, le dernier mauvais résultat est sur le test des 2 couches cachées avec 50 neurones sur la première couche et 25 sur la deuxième.





Commentaire

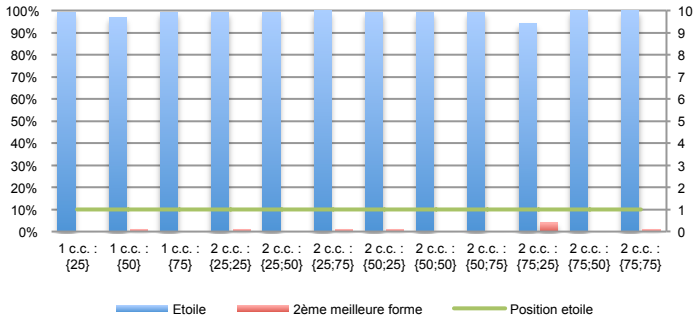
Dans les 2 cas, la forme n'a jamais été reconnue et a toujours obtenu un score inférieur à 10%, tandis que les scores par rapport autres formes est quasiment toujours supérieur à 10% et inférieur à 20%. Donc, les résultats sont de façon générale mauvais et ils ne sont finalement pas exploitables.

Conclusion

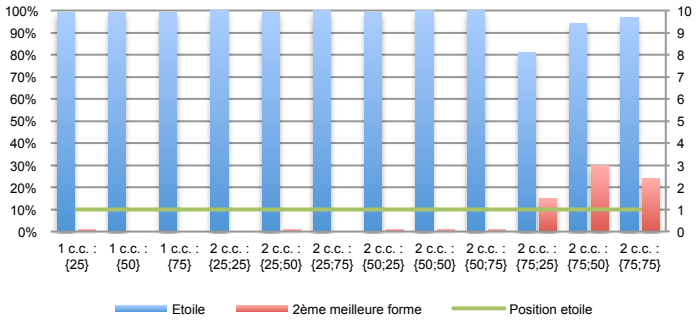
La forme en triangle obtient ici de très bons résultats avec les 10 essais pour les 10 et 20 angles, un peu moins bien sur ceux des 20 essais. Cependant, les tests avec les 50 angles sont plutôt catastrophiques et ne rentrent totalement pas dans la course des meilleurs résultats, ils permettent néanmoins de montrer que trop d'angles pour cette forme n'est pas la bonne solution. Nous allons donc sélectionner comme perceptron celui avec les 10 essais et 20 angles, car les pourcentages de reconnaissance sont quasiment parfait et les taux de reconnaissance par rapport aux autres formes sont très faibles.

Etude de la forme "Etoile"

10 Essais - 10 angles



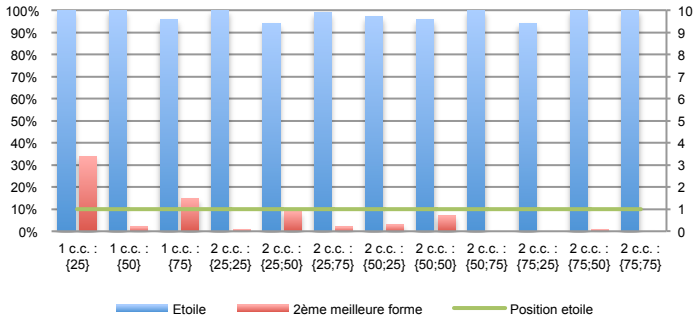
20 Essais - 10 angles



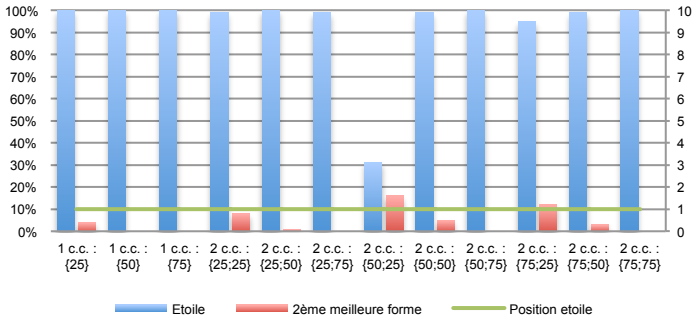
Commentaire

Le programme a dans l'ensemble parfaitement reconnu la forme. En effet tout les tests ont pratiquement 100% de taux de reconnaissance, et presque tous n'ont pas de deuxième forme à l'exception de 3 tests sur le deuxième graphique. Ces tests sont parfaits et démontrent un parfait fonctionnement du programme.

10 Essais - 20 angles

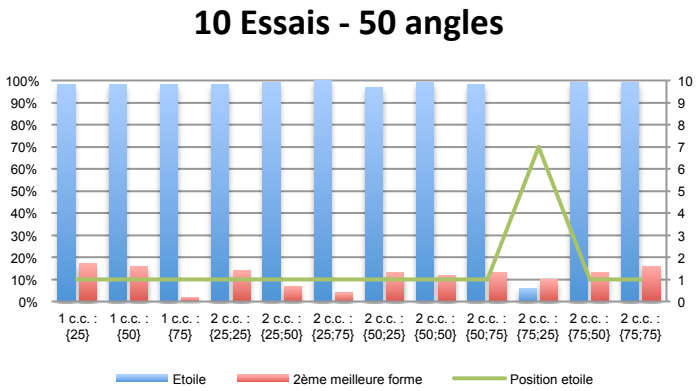


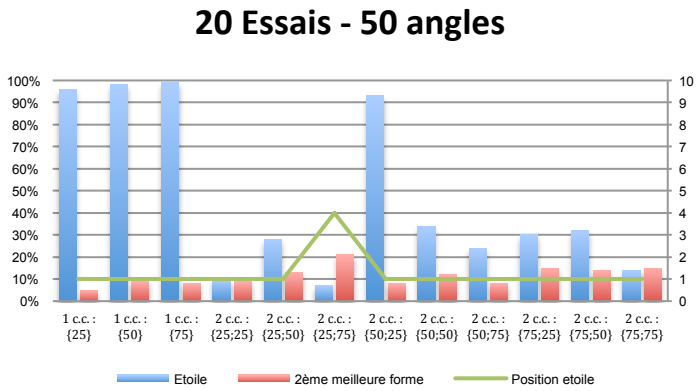
20 Essais - 20 angles



Commentaire

Les résultats renvoyés par les tests sont dans l'ensemble très bons. En effet tous les tests concluants avoisinent les 100% de taux de reconnaissance à l'exception d'un test sur le deuxième graphique qui est proche de 30%. Globalement aucune deuxième forme n'a été trouvée, à l'exception d'un cas très significatif sur le premier graphique. Malgré cela, les tests sont très bons, et reflètent un parfait fonctionnement du programme.





Commentaire

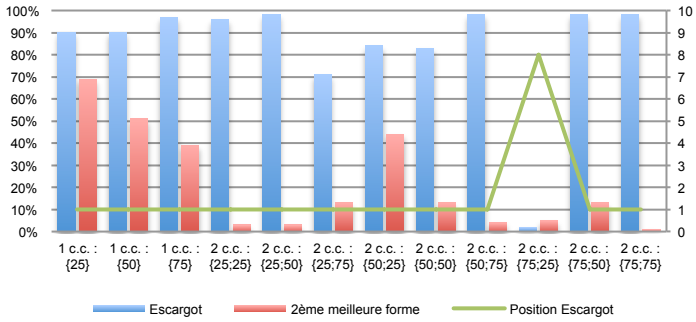
Les résultats du premier graphique sont très bons, avec des taux de reconnaissance presque à 100%, et des taux assez bas. Cependant les résultats du deuxième graphique sont mauvais. En effet, 11 tests ont été concluants, mais seuls 4 sont proches de 100% de reconnaissance. Les autres sont assez bas, et les écarts de pourcentage avec la deuxième forme la plus proche sont assez réduits. Au final, les données établies avec 2 couches cachées ne sont pas véritablement exploitables et de confiance.

Conclusion

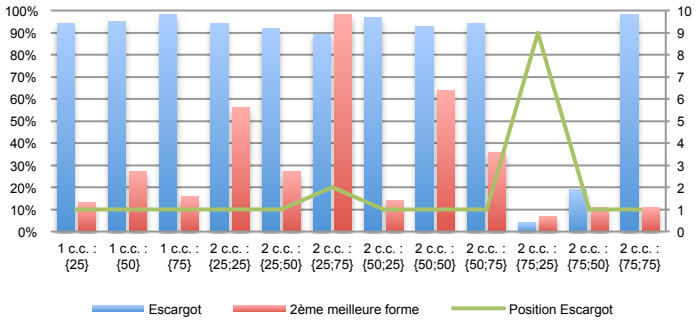
L'ensemble des tests effectués sur cette forme est très bon. Si 5 configurations sur 6 sont excellentes, c'est malgré tout "10 essais - 10 angles" qui renvoie les meilleures données, avec des taux de reconnaissance à 100% et pour ainsi dire aucune deuxième forme renvoyée. Seul le graphique "20 essais - 50 angles" met en avant des données qui sont trop mitigées et sans confiance.

Etude de la forme "Escargot"

10 Essais - 10 angles



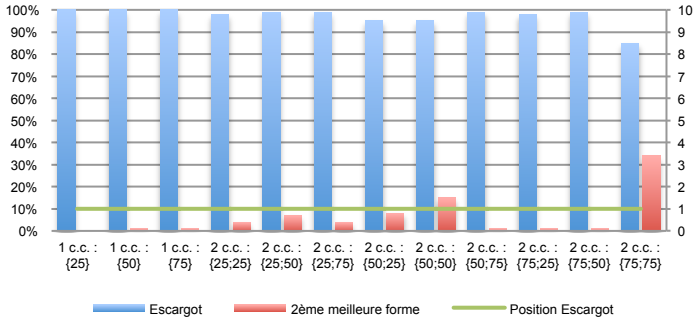
20 Essais - 10 angles



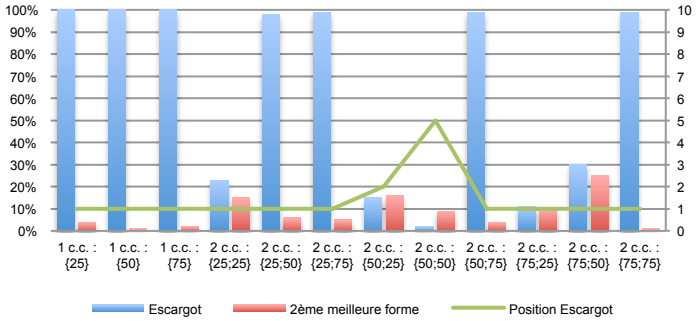
Commentaire

Dans le cas des 10 essais, la forme est quasiment tout le temps reconnue sauf dans celui des 2 couches cachées avec 75 neurones sur la première et 25 sur la deuxième, où le résultat est très mauvais. Nous obtenons également de forts taux de correspondance à d'autres formes dans plusieurs cas, comme dans celui d'une couche cachée avec 25 neurones. Dans le cas des 20 essais, nous remarquons que les résultats ne sont pas très bons dans 2 cas avec 2 couches cachées commençant avec 75 neurones. De plus, dans le cas des 2 couches cachées avec 25 neurones et 75, la forme n'est pas la première reconnues mais est seconde avec près de 90% de correspondance. Donc, on peut dire que les meilleurs résultats sont obtenus avec les 10 essais.

10 Essais - 20 angles



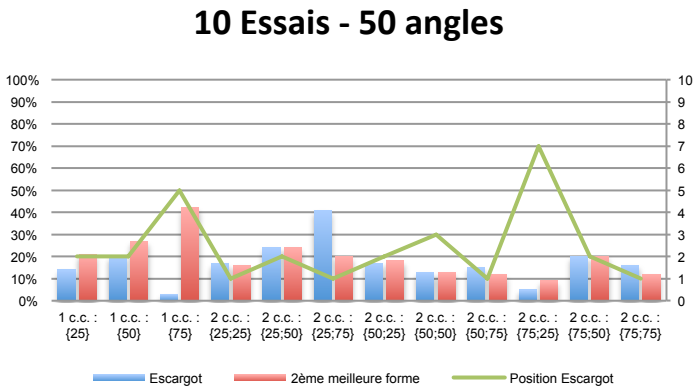
20 Essais - 20 angles

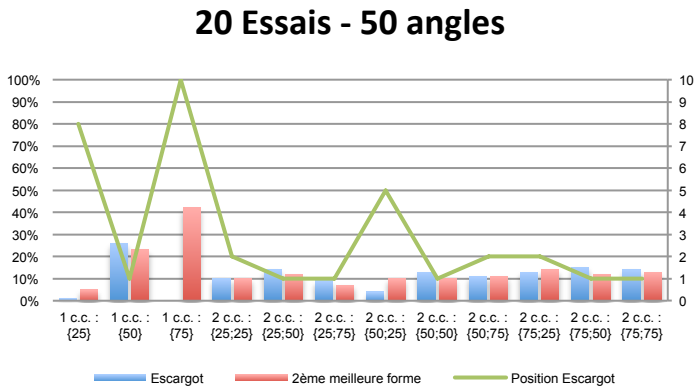


Commentaire

Le meilleur résultat est ici clairement celui des 10 essais où la forme a été trouvée à chaque fois. Dans le cas des 20 essais, les résultats sont nettement moins

bons, en effet la forme n'a été trouvée que 10 fois sur 12, mais avec 3 de ces résultats très proches des autres formes et un pourcentage inférieur à 30%. Mais les autres résultats positifs sont proches de 100%, donc très bons. Cependant les résultats, les moins bons, sont apparus avec les 2 couches cachées.





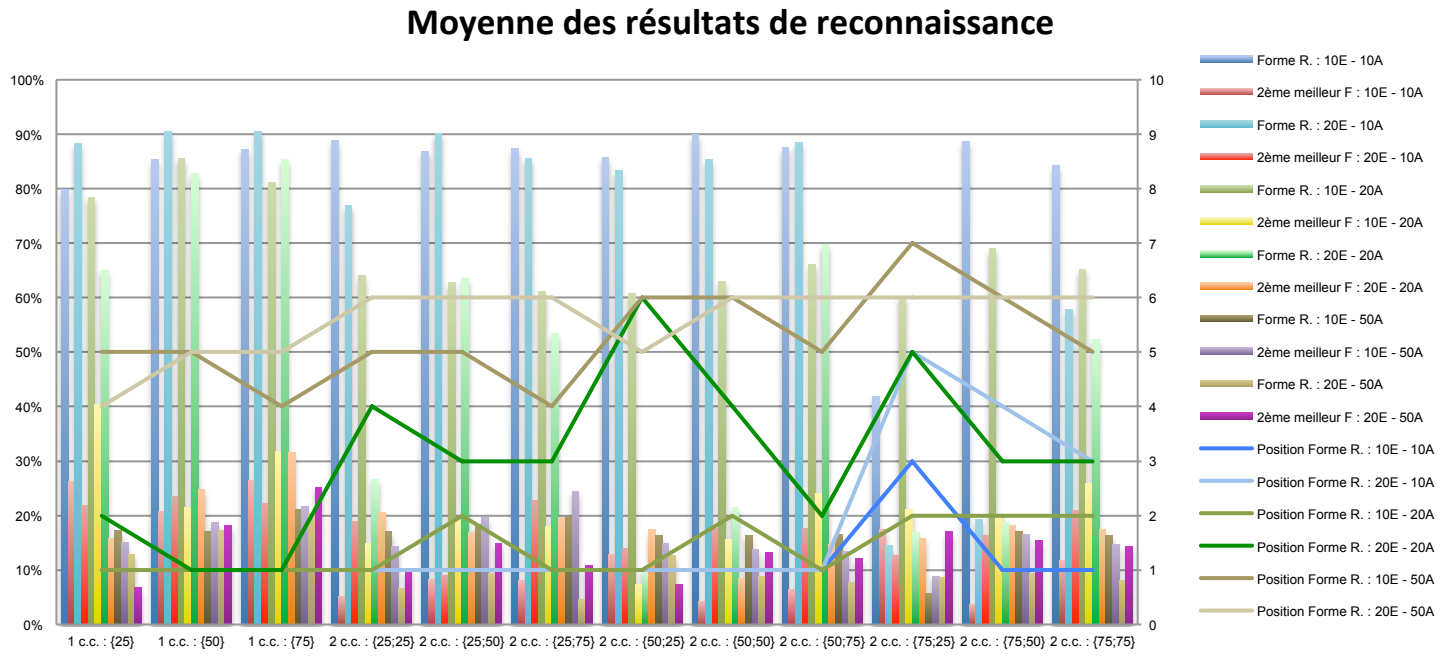
Commentaire

Les résultats pour ces tests ne sont pas très concluants. En effet, comme nous pouvons le constater, le meilleur résultat sur les 2 graphiques est de 40%. Cependant, nous pouvons remarquer que tous les pourcentages entre les formes recherchées et les deuxièmes meilleures formes sont très proches. Mais, nous remarquons que la forme recherchée est souvent placée dans les trois premières places, sauf dans 5 cas sur les cas des 10 et 20 essais.

Conclusion

On remarque que le meilleur résultat est ici le cas de 10 essais avec 20 angles. Mais nous pouvons remarquer sur le cas des 20 essais avec 20 angles, ainsi que dans les cas des 10 angles, nous obtenons de bons résultats. Cependant, les cas des 50 angles ont produit des résultats très peu satisfaisants, tout comme généralement le cas des 2 couches cachées avec 75 neurones en première couche.

Conclusion du projet



Ce graphique résume tous les tests que nous avons réalisés sur ce projet, c'est une moyenne de tous de résultats obtenus. Nous pouvons observer dans un premier temps que plus nous prenons en compte des angles et plus nous obtenons de mauvais résultats, cela est dû au fait que nous cherchons à obtenir des résultats plus précis et donc les marges d'erreurs deviennent plus faibles. Dans un second temps, nous pouvons constater que nos tests, avec 2 couches cachées où la première couche contenant plus de neurones que la deuxième, sont moins bons puisque la première couche généralise beaucoup d'informations d'entrées, qu'elle doit ensuite compresser pour les transmettre à la seconde couche, d'où la perte d'informations. Du côté des bons résultats, les graphiques montrent que la forme se place dans les premiers résultats lors des tests sur les 10 angles ce qui montre à quel point ces résultats sont stables a contrario des autres cas. De plus, nous remarquons que les résultats de ces tests sont supérieurs à 80% sauf dans le cas des 2 couches cachées avec la première couche contenant 75 neurones. L'analyse des pourcentages de reconnaissance des deuxièmes meilleures formes montre qu'elles sont toujours inférieures à 25% sur l'ensemble des moyennes faites, et qu'il

est généralement plus faible lorsque 2 couches cachées sont présentes, car le rajout d'une couche permet d'ajuster un peu mieux les poids des neurones et ainsi éviter les erreurs.

Grâce à ce graphique, nous pouvons déterminer le meilleur paramétrage des couches cachées qui est celui de 1 couche cachée avec 75 neurones avec une base de connaissance de 20 essais par forme avec plus de 90% de bonne reconnaissance. Cependant, nous remarquons que ce paramétrage a un fort taux d'erreur d'environ 20%, tandis que nous voyons que le résultat des 2 couches cachées avec 50 neurones sur les 2 couches avec les 10 essais par forme dans la base de connaissance a un résultat moyen un peu inférieur à 90% et un taux d'erreur inférieur à 5%.

Ce projet montre bien à quel point il est difficile de bien paramétrer un perceptron multi-couches, car il y a beaucoup de variables à ajuster, auxquelles s'ajoute une base de connaissances qui joue également un très grand rôle. Ce que nous voulions montrer ici est à la fois cette difficulté, mais aussi les très bons résultats que nous pouvons obtenir lorsqu'il est bien ajusté. Ainsi nous pouvons maintenant utiliser ces résultats dans un jeu vidéo pour déterminer les comportements de l'utilisateur et adapter ainsi celui des ennemis gérés par le jeu, le tout rapidement et facilement car toute la phase d'apprentissage a été réalisée en amont et il ne reste plus qu'à appliquer le perceptron durant le jeu.

Problèmes rencontrés

Durant la rédaction de ce mémoire, nous avons eu des phases de recherche très fastidieuses à réaliser car nous ne connaissions pas ce domaine d'expertise, ce qui a amené de très fortes pentes d'apprentissage. De plus, nous avons eu beaucoup de mal à trouver des références sur les algorithmes d'apprentissage directement liés aux jeux vidéo, car les développeurs ne précisent généralement pas comment ils procèdent. Cependant nous avons réussi à récupérer beaucoup d'informations sur leurs utilisations dans les cas généraux. Malgré tout, notre volonté à réussir ce mémoire et notre soif de savoir, nous a amenés à surpasser ces difficultés.

Puis, la phase de réalisation du projet a débuté, et le développement du perceptron simple couche a commencé. Il a très vite été opérationnel, ce qui n'a pas été le cas du perceptron multi-couches car nous doutions de son bon fonctionnement au vu des résultats produits. Cela nous a poussé à travailler avec le perceptron multi-couches réalisé par une autre personne car nous avions la date limite de rendu de ce mémoire qui approchait et nous avions toujours aucune base de travail. Cette solution nous a tout de même conduit à revoir à plusieurs reprises notre code de capture de forme pour le format des données d'entrée. En effet, ces données devaient être comprises entre [-1;1], or nous travaillions à la base sur des angles compris entre [0;360] et cela produisait des résultats incompréhensibles, c'est pour cela que nous avons décidé de modifier notre échelle afin qu'elle soit entre [0;1].

Pour terminer, nous avons été confrontés à un problème de gestion du temps. En effet, nous avions également beaucoup d'autres projets à réaliser simultanément à la rédaction de ce mémoire, ce qui a engendré des difficultés lors des rendus intermédiaires requis.

Apports personnels

Fabien Sacriste

Lorsque nous avons décidé d'étudier ce sujet de mémoire, j'étais très attiré par l'apprentissage numérique, mais je n'avais pratiquement aucune connaissance dans ce domaine. La tâche s'est donc révélée d'autant plus difficile car avant de pouvoir commencer à préparer mes recherches, il a fallu que j'apprenne et comprenne de quoi traitai exactement ce sujet.

Au début de mes recherches et de la phase de compréhension, j'ai longtemps hésité à changer de sujet, car j'avais beaucoup de mal à assimiler et comprendre les différents procédés, algorithmes et autres techniques. Heureusement, mon binôme m'a beaucoup aidé, et à force de travail et d'acharnement j'ai pu me mettre à niveau pour attaquer le mémoire dans de bonnes conditions.

La faible présence d'informations sur les différents types d'apprentissage dans les jeux, a été un gros problème et la production de contenu s'en est fait

ressentir, cependant l'envie et la nécessité de réussir m'ont poussé à aller de l'avant et surpasser ces difficultés. Au-delà du fait de chercher, comprendre et retranscrire des renseignements, j'ai appris énormément de choses sur les algorithmes mis en places, mais j'ai surtout été étonné d'apprendre que certains jeux que je connaissais bien les appliquaient sans que je le sache.

Si mes phases de recherches m'ont aidé à saisir l'aspect théorique d'un perceptron multi-couches, c'est malgré tout le développement de notre projet qui m'a permis de bien en cerner les subtilités et consolider mes connaissances.

Malgré la difficulté du sujet et la peur de ne pas réussir, j'ai pris beaucoup de plaisir à rédiger ce mémoire. J'ai appris énormément de choses au sujet de l'apprentissage numérique et des diverses applications que l'on pouvait en faire, mais également sur comment mieux organiser mes recherches et en synthétiser le contenu.

Guillaume Noisette

L'intelligence artificielle m'a toujours beaucoup attiré, tout comme mon binôme, c'est pour cela que nous avons choisi ce sujet. Cependant, aucun d'entre nous n'avait de notion dans ce domaine, à part les arbres de décision. La phase de recherche et de compréhension de ces algorithmes a été difficile étant donné que nous n'avions eu aucun cours sur ce sujet, c'est donc à l'aide de plusieurs documents trouvés sur internet que nous avons pu nous former. Tout ce travail de recherche m'a permis de vraiment comprendre les subtilités des réseaux de neurones.

Le projet que nous avons réalisé m'a permis de mettre en œuvre le perceptron multi-couches que nous avions étudié en première partie, car même si nous avons utilisé une librairie faite par un tiers, nous avons tenté de le faire fonctionner le nôtre. Toutefois, après avoir terminé les tests, j'ai analysé le code source de la librairie afin de comprendre nos erreurs. Iil s'est avéré que notre application ressemble beaucoup à cette librairie bien qu'elle inclue toujours des erreurs qui n'ont pas pu être totalement corrigé avant le rendu de ce mémoire. Cela prouve que nous avions vraiment bien compris le fonctionnement des perceptrons.

Grâce à ce mémoire, j'ai pu comprendre toute la difficulté liée à la mise en place d'un algorithme d'apprentissage numérique, mais surtout du défi de trouver les valeurs permettant d'obtenir les meilleurs résultats.

Conclusion du mémoire

Tout au long de ce mémoire, nous avons pu constater que l'apprentissage numérique est présent dans les jeux vidéo. En effet, plusieurs modes regroupent les algorithmes les plus utilisés tels que les arbres décisionnels et les perceptrons (simple et multi-couches) qui sont de type supervisé, ou encore les cartes de Kohonen qui sont de type non supervisé.

Ces dernières décennies, on a vu de plus en plus de jeux intégrer des intelligences artificielles ayant subi un apprentissage, certains en amont du développement tel que *Colin McRae*, avec sa base de connaissance tirée de vrais pilotes, et d'autres a posteriori avec *Forza Motorsport* qui s'adapte à la conduite du joueur. Ces 2 méthodes de gestion de l'apprentissage ont un impact direct sur la création et l'évolution du jeu, que ce soit au moment de la conception, ou que ce soit au moment où le joueur joue.

L'adaptabilité du perceptron multi-couches fait qu'il permet de traiter différents types de données dans différents contextes. En effet, il peut aussi bien être utilisé pour apprendre un XOR que pour apprendre des formes ou autres. C'est cette modularité qui nous a poussés à choisir cet algorithme pour notre projet, qui consiste à reconnaître des déplacements d'unités.

L'objectif du projet était de mettre en application un perceptron multi-couches qui serait utilisé dans un jeu. Son but principal était de pouvoir reconnaître un déplacement spécifique où l'ordinateur pourrait s'adapter et appliquer une action en conséquence (ex: prévoir l'arrivée de l'unité pour pouvoir l'éviter, ou préparer une embuscade).

Pour ce faire, nous avons mis en place différents modèles de déplacement, et nous les avons étudiés en modifiant plusieurs paramètres décisifs. Chaque forme a retournée une grande quantité de données qui, après analyse, nous a permis pour chacune d'entre elles de déterminer la configuration la plus adaptée. A l'issue des

différents tests, une configuration s'est démarquée de part ses bons taux de reconnaissance.

Avec ce projet, nous avons pu mettre en avant les difficultés à bien calibrer le perceptron grâce aux nombreux tests réalisés sur les tailles des différentes couches cachées. En effet, le nombre de paramètres étant considérable, il est apparu qu'un grand nombre de configurations était possible par forme. Effectivement, notre phase de production de données a pris plusieurs jours pour un traitement finalement peu complexe, qui plus est, dans un contexte ne faisant intervenir aucunes autres gestions (rendu graphique, physique,…). Par ailleurs, notre étude ne prend pas en compte les variations de certains paramètres tels que le taux d'apprentissage ou le taux d'inertie qui augmenteraient considérablement le nombre de configurations possibles.

Au vu des résultats obtenus, nous comprenons que le paramétrage d'un perceptron peut être complexe et fastidieux, alors qu'il est considéré comme étant le plus simple des réseaux de neurones. Cette constatation permet de mieux se rendre compte pourquoi l'apprentissage numérique n'est pas beaucoup présent dans les jeux vidéo. En effet, notre étude était basée sur la reconnaissance de forme, alors que les attentes des studios sont beaucoup plus ambitieuses. Elles le sont étant donné que les variables impactant sur une éventuelle décision sont plus nombreuses et se basent elles-mêmes sur d'autres variables. C'est le cas pour *Drivatar*, on imagine que la forme de la route n'est pas le simple facteur décisif, mais que le dénivelé, le taux de pente, le vent et la surface font partis des éléments complexes et nécessaires à prendre en compte.

Au final les développeurs doivent faire un choix entre inclure ou non un algorithme d'apprentissage numérique, tout en considérant l'impact que cela pourrait avoir sur les performances du jeu ou la qualité du game design. Au-delà du problème du temps consacré à programmer et mettre en place un apprentissage numérique, une des raisons majeure pour laquelle les développeurs n'en implémentent pas, est la limitation technique des supports sur lesquels les jeux devront fonctionner. Ce qui nous amène à nous demander si le problème de la faible utilisation de l'apprentissage numérique vient d'une complexité trop grande des algorithmes ou d'une limitation technique trop importante des machines ?

ANNEXES

Données de la forme "Carré"

Carré - 10 Essais - 10 angles												
	1 c.c. : {25}	1 c.c. : {50}	1 c.c. : {75}	2 c.c. : {25;25}	2 c.c. : {25;50}	2 c.c. : {25;75}	2 c.c. : {50;25}	2 c.c. : {50;50}	2 c.c. : {50;75}	2 c.c. : {75;25}	2 c.c. : {75;50}	2 c.c. : {75;75}
Carré	9%	17%	10%	23%	15%	45%	21%	60%	7%	7%	18%	32%
2ème meilleure forme	29%	10%	52%	27%	39%	1%	3%	2%	19%	44%	9%	9%
Position carré	2	1	2	2	2	1	1	1	2	4	1	1

Carré - 20 Essais - 10 angles												
	1 c.c. : {25}	1 c.c. : {50}	1 c.c. : {75}	2 c.c. : {25;25}	2 c.c. : {25;50}	2 c.c. : {25;75}	2 c.c. : {50;25}	2 c.c. : {50;50}	2 c.c. : {50;75}	2 c.c. : {75;25}	2 c.c. : {75;50}	2 c.c. : {75;75}
Carré	26%	35%	34%	63%	87%	70%	45%	53%	50%	5%	5%	4%
2ème meilleure forme	10%	15%	9%	6%	2%	23%	2%	13%	9%	18%	15%	4%
Position carré	1	1	1	1	1	1	1	1	1	4	7	2

Carré - 10 Essais - 20 angles												
	1 c.c. : {25}	1 c.c. : {50}	1 c.c. : {75}	2 c.c. : {25;25}	2 c.c. : {25;50}	2 c.c. : {25;75}	2 c.c. : {50;25}	2 c.c. : {50;50}	2 c.c. : {50;75}	2 c.c. : {75;25}	2 c.c. : {75;50}	2 c.c. : {75;75}
Carré	100%	97%	100%	71%	33%	98%	44%	39%	96%	97%	99%	100%
2ème meilleure forme	4%	2%	1%	0%	29%	0%	15%	17%	1%	4%	1%	0%
Position carré	1	1	1	1	1	1	1	1	1	1	1	1

Carré - 20 Essais - 20 angles												
	1 c.c. : {25}	1 c.c. : {50}	1 c.c. : {75}	2 c.c. : {25;25}	2 c.c. : {25;50}	2 c.c. : {25;75}	2 c.c. : {50;25}	2 c.c. : {50;50}	2 c.c. : {50;75}	2 c.c. : {75;25}	2 c.c. : {75;50}	2 c.c. : {75;75}
Carré	97%	100%	100%	11%	100%	1%	6%	5%	48%	5%	3%	9%
2ème meilleure forme	5%	0%	0%	29%	3%	29%	17%	22%	14%	10%	8%	9%
Position carré	1	1	1	3	1	5	9	3	1	8	5	2

Carré - 10 Essais - 50 angles												
	1 c.c. : {25}	1 c.c. : {50}	1 c.c. : {75}	2 c.c. : {25;25}	2 c.c. : {25;50}	2 c.c. : {25;75}	2 c.c. : {50;25}	2 c.c. : {50;50}	2 c.c. : {50;75}	2 c.c. : {75;25}	2 c.c. : {75;50}	2 c.c. : {75;75}
Carré	6%	6%	8%	5%	10%	6%	4%	4%	4%	4%	4%	5%
2ème meilleure forme	16%	24%	27%	17%	21%	36%	18%	13%	15%	8%	21%	14%
Position carré	8	7	3	7	4	5	7	9	7	9	7	8

Carré - 20 Essais - 50 angles												
	1 c.c. : {25}	1 c.c. : {50}	1 c.c. : {75}	2 c.c. : {25;25}	2 c.c. : {25;50}	2 c.c. : {25;75}	2 c.c. : {50;25}	2 c.c. : {50;50}	2 c.c. : {50;75}	2 c.c. : {75;25}	2 c.c. : {75;50}	2 c.c. : {75;75}
Carré	4%	7%	3%	5%	6%	0%	3%	5%	5%	3%	5%	5%
2ème meilleure forme	6%	23%	27%	10%	15%	10%	6%	13%	11%	16%	15%	14%
Position carré	3	5	3	8	9	10	6	8	9	9	9	10

Données de la forme "Cercle"

Cercle - 10 Essais - 10 angles												
	1 c.c. : {25}	1 c.c. : {50}	1 c.c. : {75}	2 c.c. : {25;25}	2 c.c. : {25;50}	2 c.c. : {25;75}	2 c.c. : {50;25}	2 c.c. : {50;50}	2 c.c. : {50;75}	2 c.c. : {75;25}	2 c.c. : {75;50}	2 c.c. : {75;75}
Cercle	97%	97%	99%	99%	100%	100%	100%	100%	100%	6%	100%	100%
2ème meilleure forme	7%	7%	3%	2%	1%	1%	1%	2%	1%	9%	0%	0%
Position cercle	1	1	1	1	1	1	1	1	1	2	1	1

Cercle - 20 Essais - 10 angles												
	1 c.c. : {25}	1 c.c. : {50}	1 c.c. : {75}	2 c.c. : {25;25}	2 c.c. : {25;50}	2 c.c. : {25;75}	2 c.c. : {50;25}	2 c.c. : {50;50}	2 c.c. : {50;75}	2 c.c. : {75;25}	2 c.c. : {75;50}	2 c.c. : {75;75}
Cercle	97%	97%	98%	23%	93%	88%	95%	92%	94%	3%	5%	3%
2ème meilleure forme	1%	3%	2%	16%	4%	6%	2%	4%	3%	10%	14%	5%
Position cercle	1	1	1	1	1	1	1	1	1	10	6	1

Cercle - 10 Essais - 20 angles												
	1 c.c. : {25}	1 c.c. : {50}	1 c.c. : {75}	2 c.c. : {25;25}	2 c.c. : {25;50}	2 c.c. : {25;75}	2 c.c. : {50;25}	2 c.c. : {50;50}	2 c.c. : {50;75}	2 c.c. : {75;25}	2 c.c. : {75;50}	2 c.c. : {75;75}
Cercle	70%	79%	36%	4%	7%	1%	11%	6%	2%	1%	2%	2%
2ème meilleure forme	100%	97%	80%	31%	16%	38%	4%	7%	47%	60%	37%	96%
Position cercle	2	2	2	3	2	3	1	4	3	3	3	2

Cercle - 20 Essais - 20 angles												
	1 c.c. : {25}	1 c.c. : {50}	1 c.c. : {75}	2 c.c. : {25;25}	2 c.c. : {25;50}	2 c.c. : {25;75}	2 c.c. : {50;25}	2 c.c. : {50;50}	2 c.c. : {50;75}	2 c.c. : {75;25}	2 c.c. : {75;50}	2 c.c. : {75;75}
Cercle	3%	29%	13%	4%	0%	0%	5%	4%	0%	6%	10%	1%
2ème meilleure forme	5%	100%	93%	21%	54%	32%	18%	4%	29%	14%	20%	28%
Position cercle	3	2	2	8	10	10	10	4	10	5	3	5

Cercle - 10 Essais - 50 angles												
	1 c.c. : {25}	1 c.c. : {50}	1 c.c. : {75}	2 c.c. : {25;25}	2 c.c. : {25;50}	2 c.c. : {25;75}	2 c.c. : {50;25}	2 c.c. : {50;50}	2 c.c. : {50;75}	2 c.c. : {75;25}	2 c.c. : {75;50}	2 c.c. : {75;75}
Cercle	7%	4%	6%	6%	11%	7%	6%	6%	6%	5%	6%	5%
2ème meilleure forme	14%	19%	19%	15%	21%	32%	16%	14%	14%	9%	18%	15%
Position cercle	8	8	4	7	3	5	7	8	6	8	8	8

Cercle - 20 Essais - 50 angles												
	1 c.c. : {25}	1 c.c. : {50}	1 c.c. : {75}	2 c.c. : {25;25}	2 c.c. : {25;50}	2 c.c. : {25;75}	2 c.c. : {50;25}	2 c.c. : {50;50}	2 c.c. : {50;75}	2 c.c. : {75;25}	2 c.c. : {75;50}	2 c.c. : {75;75}
Cercle	3%	3%	6%	5%	6%	3%	1%	4%	5%	4%	5%	5%
2ème meilleure forme	6%	21%	21%	10%	14%	10%	7%	13%	12%	16%	15%	14%
Position cercle	5	9	2	9	9	8	9	9	9	8	9	10

Données de la forme "Ligne"

Ligne - 10 Essais - 10 angles												
	1 c.c. : {25}	1 c.c. : {50}	1 c.c. : {75}	2 c.c. : {25;25}	2 c.c. : {25;50}	2 c.c. : {25;75}	2 c.c. : {50;25}	2 c.c. : {50;50}	2 c.c. : {50;75}	2 c.c. : {75;25}	2 c.c. : {75;50}	2 c.c. : {75;75}
Ligne	99%	99%	98%	100%	100%	100%	100%	100%	100%	4%	100%	100%
2ème meilleure forme	1%	1%	1%	0%	0%	0%	0%	0%	0%	5%	0%	0%
Position ligne	1	1	1	1	1	1	1	1	1	3	1	1

Ligne - 20 Essais - 10 angles												
	1 c.c. : {25}	1 c.c. : {50}	1 c.c. : {75}	2 c.c. : {25;25}	2 c.c. : {25;50}	2 c.c. : {25;75}	2 c.c. : {50;25}	2 c.c. : {50;50}	2 c.c. : {50;75}	2 c.c. : {75;25}	2 c.c. : {75;50}	2 c.c. : {75;75}
Ligne	99%	99%	99%	85%	100%	96%	99%	94%	100%	2%	3%	0%
2ème meilleure forme	0%	0%	0%	37%	19%	1%	43%	4%	47%	8%	12%	7%
Position ligne	1	1	1	1	1	1	1	1	1	10	9	10

Ligne - 10 Essais - 20 angles												
	1 c.c. : {25}	1 c.c. : {50}	1 c.c. : {75}	2 c.c. : {25;25}	2 c.c. : {25;50}	2 c.c. : {25;75}	2 c.c. : {50;25}	2 c.c. : {50;50}	2 c.c. : {50;75}	2 c.c. : {75;25}	2 c.c. : {75;50}	2 c.c. : {75;75}
Ligne	99%	97%	96%	40%	0%	27%	6%	1%	46%	48%	43%	68%
2ème meilleure forme	1%	3%	3%	8%	16%	14%	8%	7%	17%	92%	45%	43%
Position ligne	1	1	1	1	10	1	2	6	1	2	1	1

Ligne - 20 Essais - 20 angles												
	1 c.c. : {25}	1 c.c. : {50}	1 c.c. : {75}	2 c.c. : {25;25}	2 c.c. : {25;50}	2 c.c. : {25;75}	2 c.c. : {50;25}	2 c.c. : {50;50}	2 c.c. : {50;75}	2 c.c. : {75;25}	2 c.c. : {75;50}	2 c.c. : {75;75}
Ligne	5%	99%	99%	6%	0%	0%	6%	3%	7%	3%	2%	0%
2ème meilleure forme	15%	4%	16%	21%	75%	3%	19%	5%	77%	19%	11%	13%
Position ligne	3	1	1	6	10	10	9	4	2	9	5	10

Ligne - 10 Essais - 50 angles												
	1 c.c. : {25}	1 c.c. : {50}	1 c.c. : {75}	2 c.c. : {25;25}	2 c.c. : {25;50}	2 c.c. : {25;75}	2 c.c. : {50;25}	2 c.c. : {50;50}	2 c.c. : {50;75}	2 c.c. : {75;25}	2 c.c. : {75;50}	2 c.c. : {75;75}
Ligne	7%	6%	1%	6%	1%	3%	6%	7%	8%	6%	7%	5%
2ème meilleure forme	14%	18%	13%	15%	21%	30%	15%	14%	14%	9%	17%	15%
Position ligne	8	7	10	8	9	8	8	7	5	6	8	8

Ligne - 20 Essais - 50 angles												
	1 c.c. : {25}	1 c.c. : {50}	1 c.c. : {75}	2 c.c. : {25;25}	2 c.c. : {25;50}	2 c.c. : {25;75}	2 c.c. : {50;25}	2 c.c. : {50;50}	2 c.c. : {50;75}	2 c.c. : {75;25}	2 c.c. : {75;50}	2 c.c. : {75;75}
Ligne	2%	3%	0%	5%	7%	3%	5%	4%	6%	5%	5%	5%
2ème meilleure forme	7%	20%	13%	10%	15%	10%	7%	13%	13%	18%	16%	15%
Position ligne	7	9	10	9	7	8	3	9	6	7	8	10

Données de la forme "Z"

Z - 10 Essais - 10 angles												
	1 c.c. : {25}	1 c.c. : {50}	1 c.c. : {75}	2 c.c. : {25;25}	2 c.c. : {25;50}	2 c.c. : {25;75}	2 c.c. : {50;25}	2 c.c. : {50;50}	2 c.c. : {50;75}	2 c.c. : {75;25}	2 c.c. : {75;50}	2 c.c. : {75;75}
Z	99%	100%	100%	100%	100%	100%	100%	100%	100%	55%	100%	100%
2ème meilleure forme	1%	0%	1%	1%	0%	0%	0%	0%	0%	3%	0%	0%
Position Z	1	1	1	1	1	1	1	1	1	1	1	1

Z - 20 Essais - 10 angles												
	1 c.c. : {25}	1 c.c. : {50}	1 c.c. : {75}	2 c.c. : {25;25}	2 c.c. : {25;50}	2 c.c. : {25;75}	2 c.c. : {50;25}	2 c.c. : {50;50}	2 c.c. : {50;75}	2 c.c. : {75;25}	2 c.c. : {75;50}	2 c.c. : {75;75}
Z	100%	100%	100%	100%	100%	100%	100%	100%	100%	7%	9%	98%
2ème meilleure forme	0%	0%	0%	2%	1%	0%	0%	0%	0%	26%	32%	7%
Position Z	1	1	1	1	1	1	1	1	1	3	5	1

Z - 10 Essais - 20 angles												
	1 c.c. : {25}	1 c.c. : {50}	1 c.c. : {75}	2 c.c. : {25;25}	2 c.c. : {25;50}	2 c.c. : {25;75}	2 c.c. : {50;25}	2 c.c. : {50;50}	2 c.c. : {50;75}	2 c.c. : {75;25}	2 c.c. : {75;50}	2 c.c. : {75;75}
Z	12%	88%	85%	34%	96%	13%	99%	94%	46%	1%	3%	0%
2ème meilleure forme	74%	2%	95%	43%	1%	64%	14%	14%	83%	22%	64%	55%
Position Z	2	1	2	2	1	2	1	1	2	3	4	10

Z - 20 Essais - 20 angles												
	1 c.c. : {25}	1 c.c. : {50}	1 c.c. : {75}	2 c.c. : {25;25}	2 c.c. : {25;50}	2 c.c. : {25;75}	2 c.c. : {50;25}	2 c.c. : {50;50}	2 c.c. : {50;75}	2 c.c. : {75;25}	2 c.c. : {75;50}	2 c.c. : {75;75}
Z	48%	14%	46%	3%	4%	98%	6%	1%	99%	6%	4%	98%
2ème meilleure forme	6%	86%	100%	30%	1%	0%	16%	8%	1%	12%	52%	21%
Position Z	1	2	2	10	1	1	10	8	1	5	4	1

Z - 10 Essais - 50 angles												
	1 c.c. : {25}	1 c.c. : {50}	1 c.c. : {75}	2 c.c. : {25;25}	2 c.c. : {25;50}	2 c.c. : {25;75}	2 c.c. : {50;25}	2 c.c. : {50;50}	2 c.c. : {50;75}	2 c.c. : {75;25}	2 c.c. : {75;50}	2 c.c. : {75;75}
Z	8%	5%	26%	8%	1%	13%	8%	10%	7%	6%	8%	9%
2ème meilleure forme	13%	16%	33%	14%	24%	39%	14%	13%	12%	9%	16%	15%
Position Z	3	8	2	5	8	2	6	3	7	7	4	3

Z - 20 Essais - 50 angles												
	1 c.c. : {25}	1 c.c. : {50}	1 c.c. : {75}	2 c.c. : {25;25}	2 c.c. : {25;50}	2 c.c. : {25;75}	2 c.c. : {50;25}	2 c.c. : {50;50}	2 c.c. : {50;75}	2 c.c. : {75;25}	2 c.c. : {75;50}	2 c.c. : {75;75}
Z	5%	6%	5%	6%	7%	4%	4%	5%	6%	6%	6%	7%
2ème meilleure forme	7%	18%	69%	10%	16%	10%	7%	14%	13%	18%	15%	15%
Position Z	3	4	3	5	6	5	6	7	5	6	6	6

Données de la forme en "U"

U - 10 Essais - 10 angles												
	1 c.c. : {25}	1 c.c. : {50}	1 c.c. : {75}	2 c.c. : {25;25}	2 c.c. : {25;50}	2 c.c. : {25;75}	2 c.c. : {50;25}	2 c.c. : {50;50}	2 c.c. : {50;75}	2 c.c. : {75;25}	2 c.c. : {75;50}	2 c.c. : {75;75}
U	91%	83%	91%	94%	98%	91%	100%	93%	96%	5%	100%	95%
2ème meilleure forme	2%	2%	5%	5%	3%	6%	4%	0%	10%	33%	1%	9%
Position U	1	1	1	1	1	1	1	1	1	4	1	1

U - 20 Essais - 10 angles												
	1 c.c. : {25}	1 c.c. : {50}	1 c.c. : {75}	2 c.c. : {25;25}	2 c.c. : {25;50}	2 c.c. : {25;75}	2 c.c. : {50;25}	2 c.c. : {50;50}	2 c.c. : {50;75}	2 c.c. : {75;25}	2 c.c. : {75;50}	2 c.c. : {75;75}
U	99%	98%	98%	95%	99%	99%	99%	96%	98%	4%	7%	0%
2ème meilleure forme	7%	4%	6%	33%	18%	5%	3%	17%	8%	12%	12%	15%
Position U	1	1	1	1	1	1	1	1	1	7	4	10

U - 10 Essais - 20 angles												
	1 c.c. : {25}	1 c.c. : {50}	1 c.c. : {75}	2 c.c. : {25;25}	2 c.c. : {25;50}	2 c.c. : {25;75}	2 c.c. : {50;25}	2 c.c. : {50;50}	2 c.c. : {50;75}	2 c.c. : {75;25}	2 c.c. : {75;50}	2 c.c. : {75;75}
U	26%	9%	13%	9%	15%	60%	5%	18%	28%	8%	86%	44%
2ème meilleure forme	91%	41%	23%	19%	43%	23%	10%	18%	41%	9%	15%	16%
Position U	2	2	2	2	3	1	3	2	2	2	1	1

U - 20 Essais - 20 angles												
	1 c.c. : {25}	1 c.c. : {50}	1 c.c. : {75}	2 c.c. : {25;25}	2 c.c. : {25;50}	2 c.c. : {25;75}	2 c.c. : {50;25}	2 c.c. : {50;50}	2 c.c. : {50;75}	2 c.c. : {75;25}	2 c.c. : {75;50}	2 c.c. : {75;75}
U	13%	98%	100%	8%	90%	38%	7%	3%	50%	3%	1%	6%
2ème meilleure forme	12%	31%	16%	21%	5%	24%	18%	10%	4%	13%	10%	6%
Position U	1	1	1	5	1	1	6	6	1	10	7	2

U - 10 Essais - 50 angles												
	1 c.c. : {25}	1 c.c. : {50}	1 c.c. : {75}	2 c.c. : {25;25}	2 c.c. : {25;50}	2 c.c. : {25;75}	2 c.c. : {50;25}	2 c.c. : {50;50}	2 c.c. : {50;75}	2 c.c. : {75;25}	2 c.c. : {75;50}	2 c.c. : {75;75}
U	8%	7%	23%	10%	6%	12%	9%	7%	8%	6%	9%	6%
2ème meilleure forme	14%	18%	31%	16%	23%	40%	16%	14%	14%	9%	18%	15%
Position U	5	5	2	4	6	3	4	7	4	5	4	6

U - 20 Essais - 50 angles												
	1 c.c. : {25}	1 c.c. : {50}	1 c.c. : {75}	2 c.c. : {25;25}	2 c.c. : {25;50}	2 c.c. : {25;75}	2 c.c. : {50;25}	2 c.c. : {50;50}	2 c.c. : {50;75}	2 c.c. : {75;25}	2 c.c. : {75;50}	2 c.c. : {75;75}
U	4%	3%	0%	6%	7%	4%	4%	5%	5%	7%	6%	7%
2ème meilleure forme	6%	21%	21%	10%	14%	10%	6%	13%	12%	16%	15%	14%
Position U	3	9	10	7	7	5	4	8	9	5	6	7

Données de la forme"8"

8 - 10 Essais - 10 angles												
	1 c.c. : {25}	1 c.c. : {50}	1 c.c. : {75}	2 c.c. : {25;25}	2 c.c. : {25;50}	2 c.c. : {25;75}	2 c.c. : {50;25}	2 c.c. : {50;50}	2 c.c. : {50;75}	2 c.c. : {75;25}	2 c.c. : {75;50}	2 c.c. : {75;75}
8	25%	75%	81%	77%	59%	67%	55%	64%	76%	89%	71%	17%
2ème meilleure forme	66%	77%	70%	12%	35%	59%	75%	26%	30%	55%	14%	96%
Position 8	2	2	1	1	1	1	2	1	1	1	1	2

8 - 20 Essais - 10 angles												
	1 c.c. : {25}	1 c.c. : {50}	1 c.c. : {75}	2 c.c. : {25;25}	2 c.c. : {25;50}	2 c.c. : {25;75}	2 c.c. : {50;25}	2 c.c. : {50;50}	2 c.c. : {50;75}	2 c.c. : {75;25}	2 c.c. : {75;50}	2 c.c. : {75;75}
8	72%	84%	82%	15%	34%	13%	14%	26%	50%	6%	13%	99%
2ème meilleure forme	99%	95%	94%	38%	17%	92%	73%	69%	70%	9%	16%	91%
Position 8	2	2	2	3	1	2	2	2	2	6	3	1

8 - 10 Essais - 20 angles												
	1 c.c. : {25}	1 c.c. : {50}	1 c.c. : {75}	2 c.c. : {25;25}	2 c.c. : {25;50}	2 c.c. : {25;75}	2 c.c. : {50;25}	2 c.c. : {50;50}	2 c.c. : {50;75}	2 c.c. : {75;25}	2 c.c. : {75;50}	2 c.c. : {75;75}
8	77%	85%	87%	88%	92%	18%	65%	85%	52%	57%	58%	54%
2ème meilleure forme	96%	64%	98%	36%	56%	32%	5%	55%	32%	23%	36%	14%
Position 8	2	1	2	1	1	2	1	1	1	1	1	1

8 - 20 Essais - 20 angles												
	1 c.c. : {25}	1 c.c. : {50}	1 c.c. : {75}	2 c.c. : {25;25}	2 c.c. : {25;50}	2 c.c. : {25;75}	2 c.c. : {50;25}	2 c.c. : {50;50}	2 c.c. : {50;75}	2 c.c. : {75;25}	2 c.c. : {75;50}	2 c.c. : {75;75}
8	84%	88%	96%	12%	43%	4%	9%	4%	94%	8%	21%	11%
2ème meilleure forme	90%	26%	87%	23%	22%	94%	19%	9%	15%	34%	31%	91%
Position 8	2	1	1	3	1	3	4	2	1	4	2	2

8 - 10 Essais - 50 angles												
	1 c.c. : {25}	1 c.c. : {50}	1 c.c. : {75}	2 c.c. : {25;25}	2 c.c. : {25;50}	2 c.c. : {25;75}	2 c.c. : {50;25}	2 c.c. : {50;50}	2 c.c. : {50;75}	2 c.c. : {75;25}	2 c.c. : {75;50}	2 c.c. : {75;75}
8	10%	9%	30%	7%	13%	14%	6%	6%	7%	6%	6%	6%
2ème meilleure forme	16%	17%	20%	12%	23%	10%	12%	18%	14%	9%	13%	17%
Position 8	4	3	1	8	4	1	9	8	8	7	9	7

8 - 20 Essais - 50 angles												
	1 c.c. : {25}	1 c.c. : {50}	1 c.c. : {75}	2 c.c. : {25;25}	2 c.c. : {25;50}	2 c.c. : {25;75}	2 c.c. : {50;25}	2 c.c. : {50;50}	2 c.c. : {50;75}	2 c.c. : {75;25}	2 c.c. : {75;50}	2 c.c. : {75;75}
8	6%	12%	79%	8%	8%	6%	5%	8%	7%	7%	8%	7%
2ème meilleure forme	10%	14%	26%	10%	17%	10%	9%	16%	15%	19%	18%	15%
Position 8	2	3	1	4	5	3	3	4	4	4	4	7

Données de la forme "S"

S - 10 Essais - 10 angles												
	1 c.c. : {25}	1 c.c. : {50}	1 c.c. : {75}	2 c.c. : {25;25}	2 c.c. : {25;50}	2 c.c. : {25;75}	2 c.c. : {50;25}	2 c.c. : {50;50}	2 c.c. : {50;75}	2 c.c. : {75;25}	2 c.c. : {75;50}	2 c.c. : {75;75}
S	100%	100%	100%	100%	100%	100%	100%	100%	100%	82%	100%	100%
2ème meilleure forme	13%	13%	9%	1%	1%	0%	0%	0%	0%	7%	0%	1%
Position S	1	1	1	1	1	1	1	1	1	1	1	1

S - 20 Essais - 10 angles												
	1 c.c. : {25}	1 c.c. : {50}	1 c.c. : {75}	2 c.c. : {25;25}	2 c.c. : {25;50}	2 c.c. : {25;75}	2 c.c. : {50;25}	2 c.c. : {50;50}	2 c.c. : {50;75}	2 c.c. : {75;25}	2 c.c. : {75;50}	2 c.c. : {75;75}
S	100%	100%	100%	99%	100%	100%	99%	100%	100%	8%	11%	100%
2ème meilleure forme	0%	2%	2%	1%	1%	2%	1%	5%	1%	15%	13%	38%
Position S	1	1	1	1	1	1	1	1	1	2	4	1

S - 10 Essais - 20 angles												
	1 c.c. : {25}	1 c.c. : {50}	1 c.c. : {75}	2 c.c. : {25;25}	2 c.c. : {25;50}	2 c.c. : {25;75}	2 c.c. : {50;25}	2 c.c. : {50;50}	2 c.c. : {50;75}	2 c.c. : {75;25}	2 c.c. : {75;50}	2 c.c. : {75;75}
S	100%	100%	99%	97%	91%	97%	88%	95%	91%	99%	100%	99%
2ème meilleure forme	2%	0%	0%	1%	9%	3%	5%	10%	18%	1%	2%	0%
Position S	1	1	1	1	1	1	1	1	1	1	1	1

S - 20 Essais - 20 angles												
	1 c.c. : {25}	1 c.c. : {50}	1 c.c. : {75}	2 c.c. : {25;25}	2 c.c. : {25;50}	2 c.c. : {25;75}	2 c.c. : {50;25}	2 c.c. : {50;50}	2 c.c. : {50;75}	2 c.c. : {75;25}	2 c.c. : {75;50}	2 c.c. : {75;75}
S	99%	100%	100%	14%	100%	97%	9%	10%	100%	9%	13%	100%
2ème meilleure forme	13%	0%	1%	21%	0%	7%	19%	5%	1%	26%	5%	2%
Position S	1	1	1	2	1	1	3	1	1	3	1	1

S - 10 Essais - 50 angles												
	1 c.c. : {25}	1 c.c. : {50}	1 c.c. : {75}	2 c.c. : {25;25}	2 c.c. : {25;50}	2 c.c. : {25;75}	2 c.c. : {50;25}	2 c.c. : {50;50}	2 c.c. : {50;75}	2 c.c. : {75;25}	2 c.c. : {75;50}	2 c.c. : {75;75}
S	12%	13%	11%	13%	17%	2%	11%	12%	13%	9%	12%	12%
2ème meilleure forme	13%	19%	14%	13%	14%	13%	14%	14%	14%	7%	16%	14%
Position S	2	2	2	2	1	8	2	2	2	1	2	2

S - 20 Essais - 50 angles												
	1 c.c. : {25}	1 c.c. : {50}	1 c.c. : {75}	2 c.c. : {25;25}	2 c.c. : {25;50}	2 c.c. : {25;75}	2 c.c. : {50;25}	2 c.c. : {50;50}	2 c.c. : {50;75}	2 c.c. : {75;25}	2 c.c. : {75;50}	2 c.c. : {75;75}
S	8%	12%	9%	10%	9%	9%	7%	10%	7%	10%	10%	8%
2ème meilleure forme	7%	17%	5%	10%	15%	10%	5%	13%	12%	18%	16%	15%
Position S	1	2	1	3	3	2	1	3	3	3	3	4

Données de la forme "Triangle"

Triangle - 10 Essais - 10 angles												
	1 c.c. : {25}	1 c.c. : {50}	1 c.c. : {75}	2 c.c. : {25;25}	2 c.c. : {25;50}	2 c.c. : {25;75}	2 c.c. : {50;25}	2 c.c. : {50;50}	2 c.c. : {50;75}	2 c.c. : {75;25}	2 c.c. : {75;50}	2 c.c. : {75;75}
Triangle	90%	95%	97%	100%	100%	100%	99%	100%	100%	75%	100%	100%
2ème meilleure forme	75%	46%	85%	0%	0%	0%	0%	0%	0%	9%	0%	0%
Position triangle	1	1	1	1	1	1	1	1	1	1	1	1

Triangle - 20 Essais - 10 angles												
	1 c.c. : {25}	1 c.c. : {50}	1 c.c. : {75}	2 c.c. : {25;25}	2 c.c. : {25;50}	2 c.c. : {25;75}	2 c.c. : {50;25}	2 c.c. : {50;50}	2 c.c. : {50;75}	2 c.c. : {75;25}	2 c.c. : {75;50}	2 c.c. : {75;75}
Triangle	97%	98%	97%	95%	97%	100%	87%	99%	99%	25%	27%	80%
2ème meilleure forme	87%	89%	93%	0%	0%	0%	0%	0%	2%	6%	9%	7%
Position triangle	1	1	1	1	1	1	1	1	1	1	1	1

Triangle - 10 Essais - 20 angles												
	1 c.c. : {25}	1 c.c. : {50}	1 c.c. : {75}	2 c.c. : {25;25}	2 c.c. : {25;50}	2 c.c. : {25;75}	2 c.c. : {50;25}	2 c.c. : {50;50}	2 c.c. : {50;75}	2 c.c. : {75;25}	2 c.c. : {75;50}	2 c.c. : {75;75}
Triangle	100%	100%	100%	100%	100%	100%	98%	100%	100%	100%	100%	100%
2ème meilleure forme	1%	2%	1%	5%	4%	0%	2%	6%	0%	0%	0%	0%
Position triangle	1	1	1	1	1	1	1	1	1	1	1	1

Triangle - 20 Essais - 20 angles												
	1 c.c. : {25}	1 c.c. : {50}	1 c.c. : {75}	2 c.c. : {25;25}	2 c.c. : {25;50}	2 c.c. : {25;75}	2 c.c. : {50;25}	2 c.c. : {50;50}	2 c.c. : {50;75}	2 c.c. : {75;25}	2 c.c. : {75;50}	2 c.c. : {75;75}
Triangle	100%	100%	100%	87%	100%	99%	7%	83%	100%	22%	4%	100%
2ème meilleure forme	4%	0%	1%	17%	1%	2%	16%	7%	2%	8%	16%	3%
Position triangle	1	1	1	1	1	1	6	1	1	1	5	1

Triangle - 10 Essais - 50 angles												
	1 c.c. : {25}	1 c.c. : {50}	1 c.c. : {75}	2 c.c. : {25;25}	2 c.c. : {25;50}	2 c.c. : {25;75}	2 c.c. : {50;25}	2 c.c. : {50;50}	2 c.c. : {50;75}	2 c.c. : {75;25}	2 c.c. : {75;50}	2 c.c. : {75;75}
Triangle	2%	3%	5%	0%	1%	0%	0%	0%	0%	4%	0%	0%
2ème meilleure forme	12%	14%	15%	11%	18%	20%	12%	13%	12%	9%	13%	14%
Position triangle	10	10	5	10	9	10	10	10	10	10	10	10

Triangle - 20 Essais - 50 angles												
	1 c.c. : {25}	1 c.c. : {50}	1 c.c. : {75}	2 c.c. : {25;25}	2 c.c. : {25;50}	2 c.c. : {25;75}	2 c.c. : {50;25}	2 c.c. : {50;50}	2 c.c. : {50;75}	2 c.c. : {75;25}	2 c.c. : {75;50}	2 c.c. : {75;75}
Triangle	0%	2%	2%	1%	2%	1%	0%	1%	1%	1%	1%	9%
2ème meilleure forme	8%	15%	19%	10%	17%	10%	9%	16%	14%	20%	18%	14%
Position triangle	10	10	6	10	10	9	10	10	10	10	10	3

Données de la forme "Etoile"

Etoile - 10 Essais - 10 angles												
	1 c.c. : {25}	1 c.c. : {50}	1 c.c. : {75}	2 c.c. : {25;25}	2 c.c. : {25;50}	2 c.c. : {25;75}	2 c.c. : {50;25}	2 c.c. : {50;50}	2 c.c. : {50;75}	2 c.c. : {75;25}	2 c.c. : {75;50}	2 c.c. : {75;75}
Etoile	99%	97%	99%	99%	99%	100%	99%	99%	99%	94%	100%	100%
2ème meilleure forme	0%	1%	0%	1%	0%	1%	1%	0%	0%	4%	0%	1%
Position etoile	1	1	1	1	1	1	1	1	1	1	1	1

Etoile - 20 Essais - 10 angles												
	1 c.c. : {25}	1 c.c. : {50}	1 c.c. : {75}	2 c.c. : {25;25}	2 c.c. : {25;50}	2 c.c. : {25;75}	2 c.c. : {50;25}	2 c.c. : {50;50}	2 c.c. : {50;75}	2 c.c. : {75;25}	2 c.c. : {75;50}	2 c.c. : {75;75}
Etoile	99%	99%	99%	100%	99%	100%	99%	100%	100%	81%	94%	97%
2ème meilleure forme	1%	0%	0%	0%	1%	0%	1%	1%	1%	15%	30%	24%
Position etoile	1	1	1	1	1	1	1	1	1	1	1	1

Etoile - 10 Essais - 20 angles												
	1 c.c. : {25}	1 c.c. : {50}	1 c.c. : {75}	2 c.c. : {25;25}	2 c.c. : {25;50}	2 c.c. : {25;75}	2 c.c. : {50;25}	2 c.c. : {50;50}	2 c.c. : {50;75}	2 c.c. : {75;25}	2 c.c. : {75;50}	2 c.c. : {75;75}
Etoile	100%	100%	96%	100%	94%	99%	97%	96%	100%	94%	100%	100%
2ème meilleure forme	34%	2%	15%	1%	9%	2%	3%	7%	0%	0%	1%	0%
Position etoile	1	1	1	1	1	1	1	1	1	1	1	1

Etoile - 20 Essais - 20 angles												
	1 c.c. : {25}	1 c.c. : {50}	1 c.c. : {75}	2 c.c. : {25;25}	2 c.c. : {25;50}	2 c.c. : {25;75}	2 c.c. : {50;25}	2 c.c. : {50;50}	2 c.c. : {50;75}	2 c.c. : {75;25}	2 c.c. : {75;50}	2 c.c. : {75;75}
Etoile	100%	100%	100%	99%	100%	99%	31%	99%	100%	95%	99%	100%
2ème meilleure forme	4%	0%	0%	8%	1%	0%	16%	5%	0%	12%	3%	0%
Position etoile	1	1	1	1	1	1	1	1	1	1	1	1

Etoile - 10 Essais - 50 angles												
	1 c.c. : {25}	1 c.c. : {50}	1 c.c. : {75}	2 c.c. : {25;25}	2 c.c. : {25;50}	2 c.c. : {25;75}	2 c.c. : {50;25}	2 c.c. : {50;50}	2 c.c. : {50;75}	2 c.c. : {75;25}	2 c.c. : {75;50}	2 c.c. : {75;75}
Etoile	98%	98%	98%	98%	99%	100%	97%	99%	98%	6%	99%	99%
2ème meilleure forme	17%	16%	2%	14%	7%	4%	13%	12%	13%	10%	13%	16%
Position etoile	1	1	1	1	1	1	1	1	1	7	1	1

Etoile - 20 Essais - 50 angles												
	1 c.c. : {25}	1 c.c. : {50}	1 c.c. : {75}	2 c.c. : {25;25}	2 c.c. : {25;50}	2 c.c. : {25;75}	2 c.c. : {50;25}	2 c.c. : {50;50}	2 c.c. : {50;75}	2 c.c. : {75;25}	2 c.c. : {75;50}	2 c.c. : {75;75}
Etoile	96%	98%	99%	10%	28%	7%	93%	34%	24%	30%	32%	14%
2ème meilleure forme	5%	9%	8%	9%	13%	21%	8%	12%	8%	15%	14%	15%
Position etoile	1	1	1	1	1	4	1	1	1	1	1	1

Données de la forme "Escargot"

Escargot - 10 Essais - 10 angles	1 c.c. : {25}	1 c.c. : {50}	1 c.c. : {75}	2 c.c. : {25;25}	2 c.c. : {25;50}	2 c.c. : {25;75}	2 c.c. : {50;25}	2 c.c. : {50;50}	2 c.c. : {50;75}	2 c.c. : {75;25}	2 c.c. : {75;50}	2 c.c. : {75;75}
Escargot	90%	90%	97%	96%	98%	71%	84%	83%	98%	2%	98%	98%
2ème meilleure forme	69%	51%	39%	3%	3%	13%	44%	13%	4%	5%	13%	1%
Position Escargot	1	1	1	1	1	1	1	1	1	8	1	1

Escargot - 20 Essais - 10 angles	1 c.c. : {25}	1 c.c. : {50}	1 c.c. : {75}	2 c.c. : {25;25}	2 c.c. : {25;50}	2 c.c. : {25;75}	2 c.c. : {50;25}	2 c.c. : {50;50}	2 c.c. : {50;75}	2 c.c. : {75;25}	2 c.c. : {75;50}	2 c.c. : {75;75}
Escargot	94%	95%	98%	94%	92%	89%	97%	93%	94%	4%	19%	98%
2ème meilleure forme	13%	27%	16%	56%	27%	98%	14%	64%	36%	7%	11%	11%
Position Escargot	1	1	1	1	1	2	1	1	1	9	1	1

Escargot - 10 Essais - 20 angles	1 c.c. : {25}	1 c.c. : {50}	1 c.c. : {75}	2 c.c. : {25;25}	2 c.c. : {25;50}	2 c.c. : {25;75}	2 c.c. : {50;25}	2 c.c. : {50;50}	2 c.c. : {50;75}	2 c.c. : {75;25}	2 c.c. : {75;50}	2 c.c. : {75;75}
Escargot	100%	100%	100%	98%	99%	99%	95%	95%	99%	98%	99%	85%
2ème meilleure forme	0%	1%	1%	4%	7%	4%	8%	15%	1%	1%	1%	34%
Position Escargot	1	1	1	1	1	1	1	1	1	1	1	1

Escargot - 20 Essais - 20 angles	1 c.c. : {25}	1 c.c. : {50}	1 c.c. : {75}	2 c.c. : {25;25}	2 c.c. : {25;50}	2 c.c. : {25;75}	2 c.c. : {50;25}	2 c.c. : {50;50}	2 c.c. : {50;75}	2 c.c. : {75;25}	2 c.c. : {75;50}	2 c.c. : {75;75}
Escargot	100%	100%	100%	23%	98%	99%	15%	2%	99%	11%	30%	99%
2ème meilleure forme	4%	1%	2%	15%	6%	5%	16%	9%	4%	10%	25%	1%
Position Escargot	1	1	1	1	1	1	2	5	1	1	1	1

Escargot - 10 Essais - 50 angles	1 c.c. : {25}	1 c.c. : {50}	1 c.c. : {75}	2 c.c. : {25;25}	2 c.c. : {25;50}	2 c.c. : {25;75}	2 c.c. : {50;25}	2 c.c. : {50;50}	2 c.c. : {50;75}	2 c.c. : {75;25}	2 c.c. : {75;50}	2 c.c. : {75;75}
Escargot	14%	20%	3%	17%	24%	41%	17%	13%	15%	5%	20%	16%
2ème meilleure forme	21%	27%	42%	16%	24%	20%	18%	13%	12%	9%	20%	12%
Position Escargot	2	2	5	1	2	1	2	3	1	7	2	1

Escargot - 20 Essais - 50 angles	1 c.c. : {25}	1 c.c. : {50}	1 c.c. : {75}	2 c.c. : {25;25}	2 c.c. : {25;50}	2 c.c. : {25;75}	2 c.c. : {50;25}	2 c.c. : {50;50}	2 c.c. : {50;75}	2 c.c. : {75;25}	2 c.c. : {75;50}	2 c.c. : {75;75}
Escargot	1%	26%	0%	10%	14%	9%	4%	13%	11%	13%	15%	14%
2ème meilleure forme	5%	23%	42%	10%	12%	7%	10%	10%	11%	14%	12%	13%
Position Escargot	8	1	10	2	1	1	5	1	2	2	1	1

Moyenne des résultats de reconnaissance

Global - 10 Essais - 10 angles	1 c.c. : {25}	1 c.c. : {50}	1 c.c. : {75}	2 c.c. : {25;25}	2 c.c. : {25;50}	2 c.c. : {25;75}	2 c.c. : {50;25}	2 c.c. : {50;50}	2 c.c. : {50;75}	2 c.c. : {75;25}	2 c.c. : {75;50}	2 c.c. : {75;75}
Forme R. : 10E - 10A	80%	85%	87%	89%	87%	87%	86%	90%	88%	42%	89%	84%
2ème meilleur F : 10E - 10A	26%	21%	27%	5%	8%	8%	13%	4%	6%	17%	4%	12%
Position Forme R. : 10E - 10A	1	1	1	1	1	1	1	1	1	3	1	1

Global - 20 Essais - 10 angles												
Forme R. : 20E - 10A	88%	91%	91%	77%	90%	86%	83%	85%	89%	15%	19%	58%
2ème meilleur F : 20E - 10A	22%	24%	22%	19%	9%	23%	14%	18%	18%	13%	16%	21%
Position Forme R. : 20E - 10A	1	1	1	1	1	1	1	1	1	5	4	3

Global - 10 Essais - 20 angles	1 c.c. : {25}	1 c.c. : {50}	1 c.c. : {75}	2 c.c. : {25;25}	2 c.c. : {25;50}	2 c.c. : {25;75}	2 c.c. : {50;25}	2 c.c. : {50;50}	2 c.c. : {50;75}	2 c.c. : {75;25}	2 c.c. : {75;50}	2 c.c. : {75;75}
Forme R. : 10E - 20A	78%	86%	81%	64%	63%	61%	61%	63%	66%	60%	69%	65%
2ème meilleur F : 10E - 20A	40%	21%	32%	15%	19%	18%	7%	16%	24%	21%	20%	26%
Position Forme R. : 10E - 20A	1	1	1	1	2	1	1	2	1	2	2	2

Global - 20 Essais - 20 angles	1 c.c. : {25}	1 c.c. : {50}	1 c.c. : {75}	2 c.c. : {25;25}	2 c.c. : {25;50}	2 c.c. : {25;75}	2 c.c. : {50;25}	2 c.c. : {50;50}	2 c.c. : {50;75}	2 c.c. : {75;25}	2 c.c. : {75;50}	2 c.c. : {75;75}
Forme R. : 20E - 20A	65%	83%	85%	27%	64%	54%	10%	21%	70%	17%	19%	52%
2ème meilleur F : 20E - 20A	16%	25%	32%	21%	17%	20%	17%	8%	15%	16%	18%	17%
Position Forme R. : 20E - 20A	2	1	1	4	3	3	6	4	2	5	3	3

Global - 10 Essais - 50 angles	1 c.c. : {25}	1 c.c. : {50}	1 c.c. : {75}	2 c.c. : {25;25}	2 c.c. : {25;50}	2 c.c. : {25;75}	2 c.c. : {50;25}	2 c.c. : {50;50}	2 c.c. : {50;75}	2 c.c. : {75;25}	2 c.c. : {75;50}	2 c.c. : {75;75}
Forme R. : 10E - 50A	17%	17%	21%	17%	18%	20%	16%	16%	17%	6%	17%	16%
2ème meilleur F : 10E - 50A	15%	19%	22%	14%	20%	24%	15%	14%	13%	9%	17%	15%
Position Forme R. : 10E - 50A	5	5	4	5	5	4	6	6	5	7	6	5

Global - 20 Essais - 50 angles	1 c.c. : {25}	1 c.c. : {50}	1 c.c. : {75}	2 c.c. : {25;25}	2 c.c. : {25;50}	2 c.c. : {25;75}	2 c.c. : {50;25}	2 c.c. : {50;50}	2 c.c. : {50;75}	2 c.c. : {75;25}	2 c.c. : {75;50}	2 c.c. : {75;75}
Forme R. : 20E - 50A	13%	17%	20%	7%	9%	5%	13%	9%	8%	9%	9%	8%
2ème meilleur F : 20E - 50A	7%	18%	25%	10%	15%	11%	7%	13%	12%	17%	15%	14%
Position Forme R. : 20E - 50A	4	5	5	6	6	6	5	6	6	6	6	6

Exemple de code c++ : Perceptron simple couche

Application pour apprendre la fonction "ET"

```
#include <iostream>

#ifdef USE_SIMPLE

using namespace std;

#define NUMBER_SIZE_VECTOR 3 // 2 parameters of vector and the
through
#define NUMBER_ENTRIES          4
#define LEARNING_RATE      0.1

typedef struct {
    int data[NUMBER_SIZE_VECTOR];
    int result;
} TrainingStruct;

bool threshold(double value)
{
    return value > 0.5 ? 1 : 0;
}

int main (int argc, char *argv[])
{
    TrainingStruct trainingData[NUMBER_ENTRIES] =
    {
        {
            {0, 0, 1},
            0
        },
        {
            {0, 1, 1},
            0
        },
        {
            {1, 0, 1},
            0
        },
        {
            {1, 1, 1},
            1
        }
    };
    double weight[NUMBER_SIZE_VECTOR] = { 0, 0, 0 };
    bool continueLearning, Hw;
    double sumWX;
```

```
    do
    {
        continueLearning = false;
      for (int i = 0; i < NUMBER_ENTRIES; ++i)
      {
          TrainingStruct * training = &trainingData[i];

            // Calculate w * x
            sumWX = 0;
            for(int j = 0; j < NUMBER_SIZE_VECTOR; ++j)
                  sumWX += training->data[j] * weight[j];

            // Apply the function threshold
            Hw = threshold(sumWX);

            // Apply the modification of the weight
            for (int j = 0; j < NUMBER_SIZE_VECTOR; ++j)
                  weight[j] = weight[j] + LEARNING_RATE * (training-
>result - Hw) * training->data[j];

            // Verify the end of the learning
            if (training->result - Hw != 0)
                  continueLearning = true;
      }
    } while (continueLearning);

    // Display the weight
    for (int i = 0; i < NUMBER_SIZE_VECTOR; ++i)
      cout << "weight[" << i << "] = " << weight[i] << endl;

    return 0;
}

#endif
```

BIBLIOGRAPHIE

http://fr.wikipedia.org/wiki/R%C3%A9seau_de_neurones_artificiels
http://fr.wikipedia.org/wiki/Apprentissage_automatique
http://www.grappa.univ-lille3.fr/polys/apprentissage/sortie005.html
http://wikistat.fr/pdf/st-m-app-rn.pdf
http://www.trop.uha.fr/pdf/cours-wira.pdf
http://olivier.teytaud.pagesperso-orange.fr/publis/serpilliere.pdf
Apprentissage :
http://www-igm.univ-mlv.fr/~dr/XPOSE2002/Neurones/index.php?rubrique=Apprentissage
http://eric.univ-lyon2.fr/~ricco/cours/slides/reseaux_neurones_perceptron.pdf
http://fr.wikipedia.org/wiki/Carte_auto_adaptative
Apprentissage numérique :
http://veille-techno.blogs.ec-nantes.fr/index.php/2013/02/13/apprentissage-automatique-et-reseaux-de-neurones/
http://www.youtube.com/playlist?list=PL6Xpj9I5qXYGhsvMWM53ZLfwUInzvYWsm
http://www.researchgate.net/publication/228933107_Exploiting_the_fascination_Video_games_in_machine_learning_research_and_education/file/79e4151290bcba996c.pdf
http://videolectures.net/mlss05au_graepel_mlg/
http://research.microsoft.com/en-us/events/2011summerschool/jqcandela2011.pdf
http://www.cse.lehigh.edu/~munoz/CSE497/classes/MeganNeural.ppt
http://webdocs.cs.ualberta.ca/~sutton/book/ebook/node109.html
http://www.mlplatform.nl/what-is-machine-learning/
http://videolectures.net/mlss05au_graepel_mlg/
http://thinkingmachineblog.net/use-of-machine-learning-in-video-games/
http://en.wikipedia.org/wiki/Decision_tree_learning
Apprentissage supervisé :
http://www.aihorizon.com/essays/generalai/supervised_unsupervised_machine_learning.htm
Neural network:
http://gamingbolt.com/forza-motorsport-5-wiki
http://research.microsoft.com/en-us/projects/drivatar/
http://www.ift.ulaval.ca/~lamontagne/ift17587/projets%20pass%C3%A9s%20GameAI.pdf
http://www.cplusplus.com/forum/lounge/105609/
http://www.idt.mdh.se/kurser/ct3340/ht11/MINICONFERENCE/FinalPapers/ircse11_submission_5.pdf
http://gameai.com/
http://i.imgur.com/dx7sVXj.jpg
Arbre décisionnel:
http://www.youtube.com/watch?v=G29LTOkZ8RA&feature=youtube_gdata
http://web.cs.wpi.edu/~rich/courses/imgd4000-d09/lectures/B-AI.pdf
http://www.grappa.univ-lille3.fr/polys/apprentissage/sortie004.html
http://fr.wikipedia.org/wiki/Arbre_de_d%C3%A9cision
http://web.cs.du.edu/~sturtevant/ai-s11/Lecture03.pdf
http://www.nuddz.com/questions/Comment-fonctionne-l-algorithme-de-Akinator/541
Carte de Kohonen
http://fr.wikipedia.org/wiki/Carte_auto_adaptative#Principe
http://en.wikipedia.org/wiki/Self-organizing_map
http://mimosa.cereq.fr/rousset/M2S3_octobre.pdf
http://eric.univ-lyon2.fr/~rias2006/presentations/VincentLemaire.pdf
http://beefchunk.com/documentation/ia/cours_IA/node101.html

http://asi.insa-rouen.fr/enseignants/~scanu/RdF/TP/Kohonen.htm
http://mnemstudio.org/neural-networks-kohonen-self-organizing-maps.htm
http://pro.chemist.online.fr/cours/koho1.htm
http://www.timotheecour.com/papers/vision_system.pdf
Classifieur linéaire
http://fr.wikipedia.org/wiki/Classifieur_lin%C3%A9aire
http://pageperso.lif.univ-mrs.fr/~guillaume.stempfel/Memoire.pdf
http://samifis.irisib.be/2011/01/07/classifieurs-lineaires-svm/
http://dspace.univ-tlemcen.dz/bitstream/112/322/11/ChapitreII.pdf
Perceptron
http://fr.wikipedia.org/wiki/Perceptron
http://wcours.gel.ulaval.ca/2010/a/GIF4101/default/5notes/ar-sem10-pmc.pdf
http://en.wikipedia.org/wiki/Multilayer_perceptron
http://fabien.tschirhart.free.fr/images/Docs/memoire_V129.pdf
http://wcours.gel.ulaval.ca/2009/h/19968/default/5notes/NotesDeCours/RetroPerceptron.pdf
http://themarvinproject.free.fr/final/node3.html#SECTION03400000000000000000
http://www.generation5.org/content/2001/hannan.asp
http://www.ai-junkie.com/misc/hannan/hannan.html
https://www.lri.fr/~antoine/Courses/ENSTA/Tr-ensta-nnx9.pdf
http://matlabgeeks.com/tips-tutorials/neural-networks-a-multilayer-perceptron-in-matlab/
Rétropropagation du gradient de l'erreur
http://en.wikipedia.org/wiki/Backpropagation
http://www.iro.umontreal.ca/~bengioy/ift6266/H12/html/gradient_fr.html#le-gradient
http://rfia2012.files.wordpress.com/2012/02/descente-stochastique.pdf
http://en.wikipedia.org/wiki/Rprop
http://www.heatonresearch.com/wiki/Resilient_Propagation
Autres:
http://code.google.com/p/opennero/wiki/NeroMod
Projet
http://www.ifi.auf.org/site_data/rapports/tpe-promo10/tipe-pham_quang_dung.pdf
http://www.sylbarth.com/mlp.php