Initiation à la recherche
Apprentissage du comportement d'un bras robotique

Maix Gaëtan Li Shibo

26 mai 2011

Table des matières

Résumé

Université : Henri Poincaré, Nancy1
Contexte : Initiation à la recherche
Encadrants : N.Rougier et Y.Boniface

1 Introduction sur les réseaux de neurones

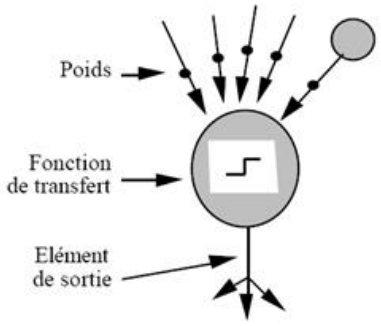
Les réseaux de neurones artificiels s'appuient sur le modèle du cerveau humain et tendent à imiter son comportement. Cependant le cerveau et sa complexité lui procure une capacité de calcul énorme en comparaison à la puissance des machines et des modèles actuelles. Plus particulièrement le processus d'apprentissage du cerveau est ainsi très développée, nous verrons par la suite comment il s'applique aux réseaux de neurones artificiels.

1.1 Définition

Definition 1. *Les réseaux de neurones artificiels sont des réseaux fortement connectés de processeurs élémentaires fonctionnant en parallèle. Chaque processeur élémentaire (l'équivalent du neurone humain) calcule une sortie unique sur la base des informations qu'il reçoit (ce que l'ont doit apprendre). Toute structure hiérarchique de réseaux est évidemment un réseau[3].*

1.2 Structure d'un neurone artificiel

Les réseaux de neurones permettent d'apprendre des fonctions non-linéaires à partir de plusieurs exemples de la fonction recherchée. Chaque neurone du réseau possède des poids qui représentent les connaissances du réseau. Ces poids sont modifiés à partir des entrées présentées grâce à la fonction de transfert, ce qui permet de modifier la réponse du réseau en fonction d'une stimulation (dans notre cas la stimulation représente une nouvelle connaissance/donnée à apprendre).

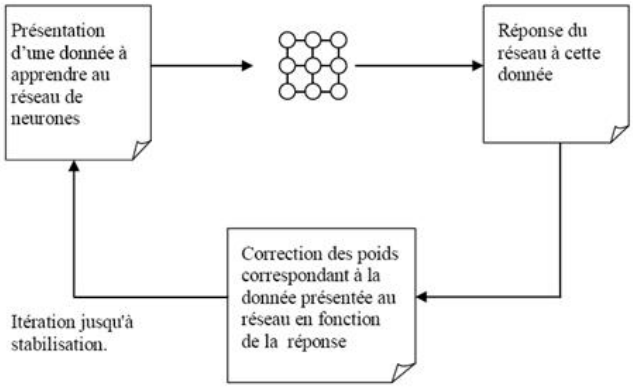


Structure d'un neurone artificiel[3]

1.3 Apprentissage d'un réseau de neurones

L'apprentissage est un processus itératif. A chaque itération on présente une donnée (ou plusieurs) à apprendre pour le réseau. Celui-ci "compare" la représentation (les poids) qu'il a de cette donnée avec la donnée réelle et ajuste les poids des neurones concernés en fonction de celle-ci.

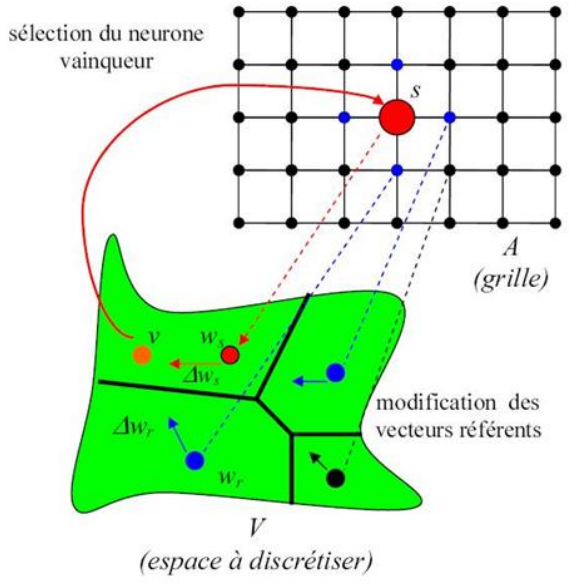
Plus l'itération avance, moins le réseau apprend. En effet au début du processus, le réseau apprend « beaucoup »(il connaît mal les données), puis il apprend de moins en moins (il converge vers la solution à apprendre et fait donc des ajustements de plus en plus petit) pour finir par se stabiliser. Il existe plusieurs algorithmes, nous expliquerons en détails le fonctionnement de notre réseau par la suite.



Principe général d'apprentissage

2 Kohonen Map

Les cartes de Kohonen appartiennent à la famille des cartes auto-organisatrices (SOM). Celles-ci permettent une discrétisation de l'espace d'entrée. Elles sont représentées par une grille régulière (exemple : matrice), où chaque nœud représente un neurone. Elles peuvent être de plusieurs dimensions (1D, 2D...).
A partir de cette structure, on définit les relations de voisinage entre les neurones. Cette propriété (le voisinage) permet d'obtenir un apprentissage assez rapide et stable. A la fin de l'apprentissage les poids d'un neurone associé à une zone de l'espace d'entrées doivent tendre vers les valeurs représentatives de celle-ci.



Exemple de discrétisation d'un espace avec une SOM[4]

2.1 Self-Organizing Map (SOM)

2.1.1 Principe

Après une initialisation aléatoire des valeurs de chaque neurones on soumet une à une les données à la carte auto adaptative. Selon les valeurs des neurones, il y en a un qui répondra le mieux au stimulus. Celui dont la valeur sera la plus proche de la donnée présentée. Alors ce neurone sera gratifié d'un changement de valeur pour qu'il réponde encore mieux à un autre stimulus de même nature que le précédent.
Par là même, on gratifie aussi un peu aussi les neurones voisins du gagnant avec un facteur multiplicatif du gain inférieur à un. Ainsi, c'est toute la région de la carte autour du neurone gagnant qui se spécialise.
En fin d'algorithme, lorsque les neurones ne bougent plus, ou très peu, à chaque itération, la carte auto organisatrice recouvre toute la topologie des données.[4]

Algorithm 1 SOM[1][2]

1: *Initialisation* Construction de données à apprendre
2: **for** $i = 0$ **to** *stabilisation* **do**
3: *choix* : choix d'une donnée au hasard
4: *vainqueur* : choix d'un neurone vainqueur suivant le critère désiré
5: $modification$: mise à jour des poids des neurones suivant une fonction de voisinage h_δ
6: **end for**

2.1.2 fontion de voisinage h_δ

Soit v le vecteur de poids de la donnée tirée au hasard, la mise à jour des poids w_i d'un neurone est définie tel que

$$w_i = w_i + \delta wi \tag{2.1}$$

$$\delta wi = \epsilon(t) h_\delta(t, d)(v - w_i) \tag{2.2}$$

avec t représentant l'itération courante, d la distance topologique (dans le réseau) entre le $neurore_i$ et $neuroneVainqueur$ et h_δ de la forme

$$h_\delta(t, d) = e^- \frac{d^2}{2\delta(t)^2} \tag{2.3}$$

avec $\delta(t)$ de la forme

$$\delta(t) = \delta_i (\frac{\delta_f}{\delta_i})^{\frac{t}{t_f}} \tag{2.4}$$

et $\epsilon(t)$ de la forme

$$\epsilon(t) = \epsilon_i(\frac{\epsilon_f}{\epsilon_i})^{\frac{t}{t_f}} \tag{2.5}$$

où ϵ_i et ϵ_f représentent le taux d'apprentissage initial et final et δ_i et δ_f représentent le taux d'apprentissage du voisinage initial et final.
En règle général, on a $\epsilon_f << \epsilon_i$ et $\delta_f << \delta_i$ (on apprend de moins en mois $\rightarrow$ stabilisation).

2.2 Dynamic Self-Organizing Map (DSOM)

L'algorithme DSOM permet un meilleur apprentissage lorsque la distribution des données varie au cours du temps (dynamique).
Pour cela on modifie la fonction de voisinage.
A la différence de SOM, lorsqu'un neurone vainqueur est considéré comme assez proche de la donnée (on considère que la donnée est suffisament apprise) alors les autres neurones du réseau n'apprennent pas cette donnée.

Algorithm 2 DSOM[1][2]

1: *Initialisation* Construction de données à apprendre
2: **for** $i = 0$ **to** *stabilisation* **do**
3: *choix* : choix d'une donnée au hasard
4: *vainqueur* : choix d'un neurone vainqueur suivant le critère désiré
5: $modification$: mise à jour des poids des neurones suivant une fonction de voisinage hn_μ
6: **end for**

2.2.1 fontion de voisinage hn_μ

Avec DSOM, les paramètres $\epsilon_i, \epsilon_f, \delta_i, \delta_f$ ne sont plus utilisés. A la place on utilise un paramètre μ, l'elasticité. Soit v le vecteur de poids de la donnée tirée au hasard, la mise à jour des poids w_i d'un neurone est définie tel que

$$w_i = w_i + \delta wi \tag{2.6}$$

$$\delta wi = d_r hn_\mu(d, d_r)(v - w_i) \tag{2.7}$$

avec t représentant l'itération courante, d_r la distance réelle entre le $neurore_i$ et $neuroneVainqueur$ (distance entre le point x, y et x', y') et hn_μ de la forme

$$hn_\mu(d, d_r) = e^-(\frac{1}{\mu^2}\frac{d^2}{d_r^2}) \tag{2.8}$$

où μ et le paramètre d'élasticité.(Si $d_r = 0$ alors $hn_\mu = 0$).

3 Travail effectué

3.1 Implémentation en Java

3.1.1 Présentation des classes

Data.java

Cette classe est une structure de donnée représentant les données (exemples) qui seront soumis au réseau de neurones pour son apprentissage. Elle comporte donc les valeurs (poids) de chaque donnée.
Dans notre cas les poids des données représentent les coordonnées X et Y de la donnée.

Neurone.java

Cette classe est une structure de donnée représentant un neurone unique. Elle contient donc les poids des neurones (initialisés aléatoirement par le constructeur), leur fonction de transfert (permettant de modifier les poids du neurone). Dans notre cas les poids des neurones représentent les coordonnées X et Y du neurone.

Neural Network.java

C'est la classe principale représentant le réseau de neurone et sa topologie (une matrice dans notre cas). Cette classe contient tous les paramètres du réseau (taux d'apprentissage, élasticité, nombre d'itération, nombre de données, etc...).
Elle contient aussi l'implémentation (méthode run()) des algorithmes SOM et DSOM décrite ci-dessous en détail.

3.1.2 Implémentation des algorithmes SOM et DSOM

Fonction de voisinage SOM

```
//fonction de voisinage SOM
private double h(int iter, Neurone neurone, Neurone currentBest) {
  return Math.exp(-1.0*distance(neurone,currentBest)/sigma_f(iter));
}
//sigma_f
private double sigma_f(int iter) {
    return 2*Math.pow((sigma_init*Math.pow((sigma_final /sigma_init)↩
        ,((double)iter/(double)max_iter))),2);
}

//distance entre deux neurones (voisinage)
private double distance(Neurone n1, Neurone currentBest) {
    return Math.pow(distance_voisins(currentBest,n1),2);
}//distance_voisins est la distance topologique sur le reseau
```

Fonction de changement du poids d'un neurone (SOM)

```
public void setPoids(double e, double h, Data data,boolean mode) {
  double wi;
  for(int i=0;i<poids.size();i++){
    wi = poids.get(i);
    wi = wi + e*h*(data.getPoids().get(i) - wi); //SOM
    poids.set(i, new Float(wi));
  }
}
```

Avec e = epsilon en fonction de l'itération courante définit comme :

```
private double epsilon_f(int iter){
  return e_init*Math.pow((e_final /e_init),((double)iter/(double)↩
      max_iter));
}
```

Fonction de voisinage DSOM

```
//fonction de voisinage DSOM
private double hn(Neurone i,Neurone s, Data v) {
  return Math.exp((-1.0/Math.pow(elasticity,2))*(distance(i,s)/(Math.↩
      pow(s.getDistanceXY(v), 2))));
}

//distance euclidienne
public double getDistanceXY(Data data) {
  double res = 0.0;
  for(int i=0;i<poids.size();i++){
    res +=Math.pow(Math.abs((poids.get(i)data.getPoids().get(i))),2);
  }
  return Math.sqrt(res)/(double)echelle;
}
```

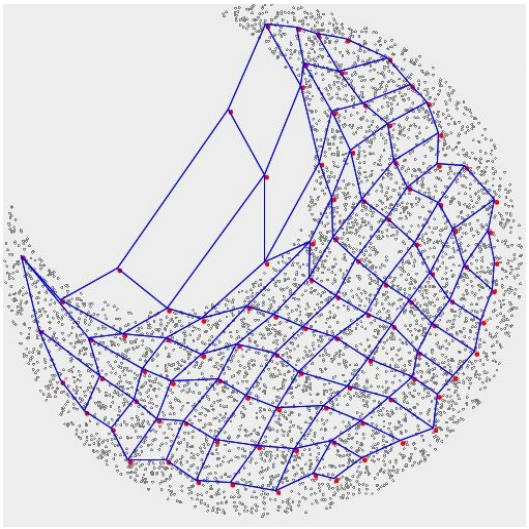
Fonction de changement du poids d'un neurone (DSOM)

```
public void setPoids(double e, double h, Data data,boolean mode) {
  double wi;
  for(int i=0;i<poids.size();i++){
    wi = poids.get(i);
    wi = wi+this.getDistanceXY(data)*h*(data.getPoids().get(i) - wi); ↩
        //DSOM
  }
  poids.set(i, new Float(wi));
}
```

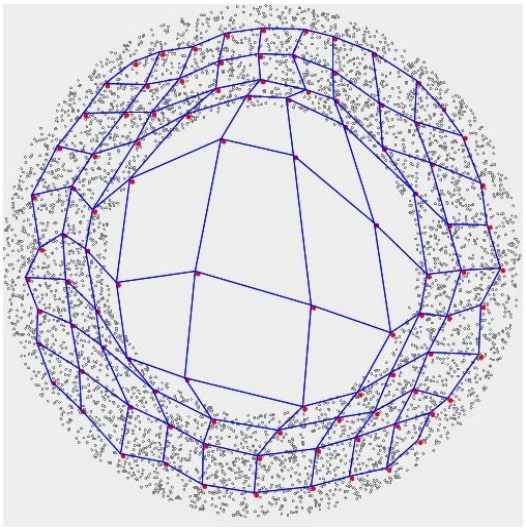
3.2 Exemples d'apprentissages

Des espaces de données divers (plus ou moins uniformes) on été générés afin de tester les algorithmes. En voici quelques exemples ci-dessous.

3.2.1 Exemples d'apprentissages SOM

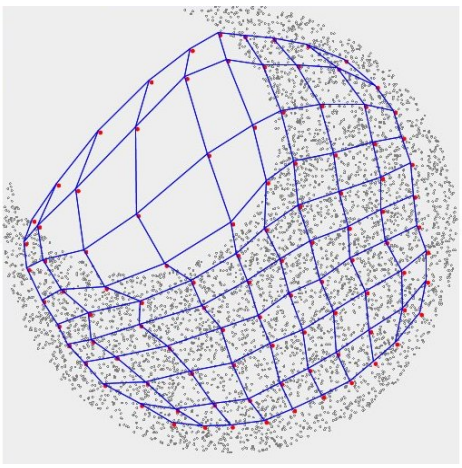


(A) SOM lune

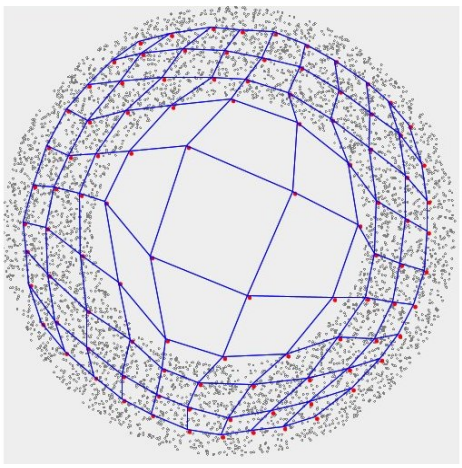


(B) SOM anneau

3.2.2 Exemples d'apprentissages DSOM



(A) DSOM lune

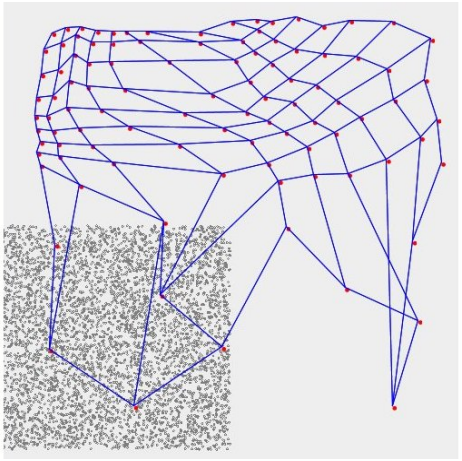


(B) DSOM anneau

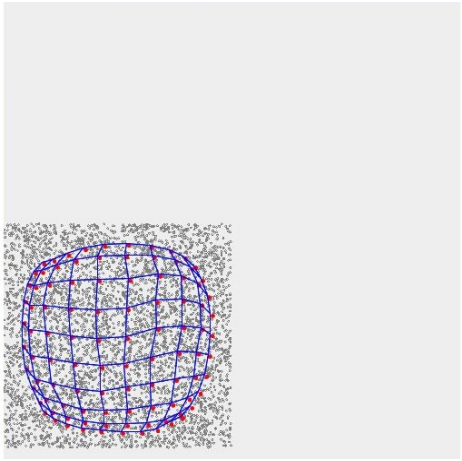
3.2.3 Différence entre SOM et DSOM avec une distribution dynamique

L'espace de la distribution varie au cours des itérations dans le sens

$$NO \rightarrow NE \rightarrow SE \rightarrow SO \tag{3.1}$$



(A) distribution dynamique avec SOM



(B) distribution dynamique avec DSOM

4 Application au bras robotique

4.1 Théorie

On dispose d'un bras robotique à deux degrés de libertés composés par des segments l_1 et l_2 (longueur du bras et de l'avant bras).
Le but de l'application est d'utiliser le réseau de neurone développé afin de discrétiser l'espace et permettre au robot d'associer à une position x, y les angles θ_1 (premier degré de liberté) et θ_2 (second degré de liberté).

Pour cela, on va créer des données à apprendre en tirant aléatoirement des angles θ_1 et θ_2 compris entre $\left[-\frac{\pi}{2}; \frac{\pi}{2}\right]$ et en calculant les positions réelles x_r, y_r associées par rapport à un point de référence (départ du segemnt l_1).

$$x_r = reference_x + l_1 cos(\theta_1) + l_2 cos(\theta_2) \tag{4.1}$$
$$y_r = reference_y - l_1 sin(\theta_1) - l_2 sin(\theta_2) \tag{4.2}$$

Les neurones sont composés de poids x, y et θ_1, θ_2 tirés aléatoirement.
Lors de l'apprentissage, deux cartes SOM ou DSOM apprennent en parallèle. Une carte est dédiée aux coordonnées x, y et l'autre aux angles θ_1, θ_2. On choisit le neurone vainqueur en prenant la distance minimale par rapport à la donnée, définie comme suit :

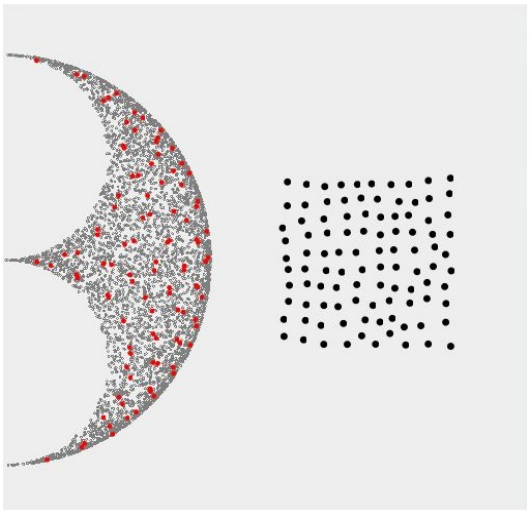
$$\frac{1}{2} distanceSOMxy + \frac{1}{2} distanceSOM\theta_1\theta_2 \tag{4.3}$$

On réitère l'opération jusqu'à stabilisation.

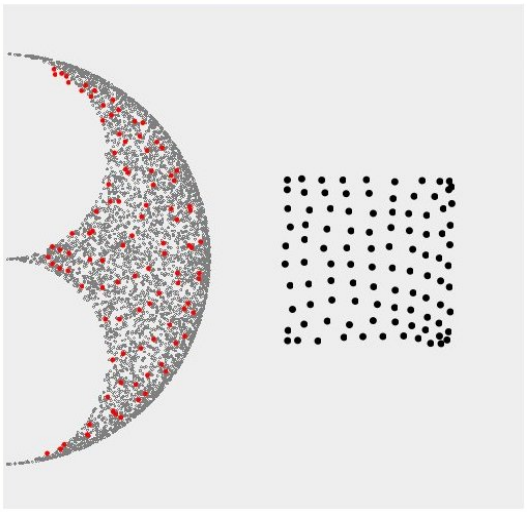
4.1.1 Exemples d'apprentissages théoriques du bras

Grâce à cet apprentissage, on obtient une association entre les coordonnées x, y des neurones et les angles θ_1, θ_2 à appliquer au bras pour atteindre cette position.
De plus on observe en rouge le SOM/DSOM associé aux coordonnées x, y et en noir le SOM/DSOM associé aux angles θ_1, θ_2. Ce denier est bien uniforme comme attendu (on a tiré les angles au hasard de manière uniforme pour créer les données).

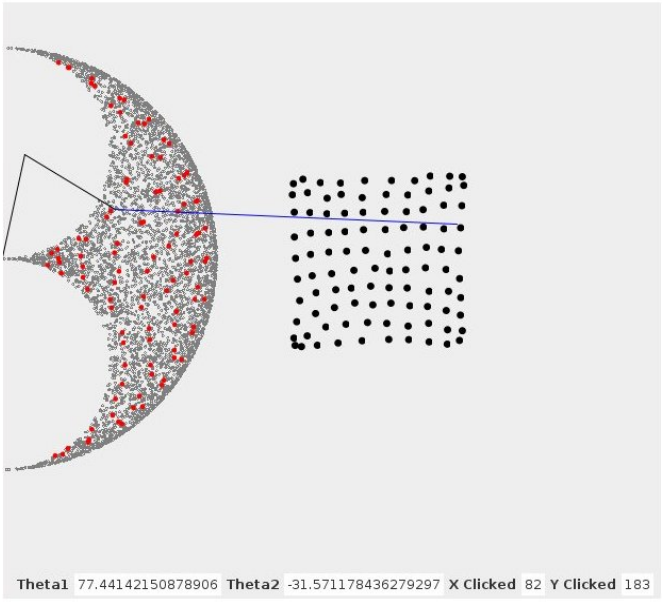
On peut alors utilisé le réseau de neurone pour associé une position x, y à des angles θ_1, θ_2 en choisissant le neurone le plus proche (gagnant) des coordonnées x, y choisis.



(A) SOM robot



(B) DSOM robot



En bleu l'association entre x, y (cliqué) et θ_1, θ_2
En noir les segments l_1 et l_2 du bras robotique

4.1.2 Apprentissages alternatifs avec DSOM

Les apprentissages suivant ne sont réalisable qu'avec DSOM grâce à son caractère dynamique.

Exploration par balbutiements moteurs

Même principe que précédemment excepté que les données ne sont pas créés à l'avance.

Algorithm 3 DSOM Moteur[1][2]

for $i = 0$ **to** k **do**
  choix : tirage de deux angles au hasard
  calcul : calcul de la donée réelle δ avec ces deux angles
  vainqueur : choix d'un neurone vainqueur
  $modification$: mise à jour des poids des neurones suivant une fonction de voisinage hn_μ en apprenant δ
end for

Exploration par balbutiements rétinien

On simule une rétine (un plan 2D, une webcam en pratique)

Algorithm 4 DSOM Retine[1][2]

for $i = 0$ **to** k **do**
  choix : on prend un point sur la rétine de façon aléatoire
  vainqueur : choix du neurone n le plus proche dans l'espace rétinien
  calcul : à l'aide des coordonnées motrices de ce neurone n, on calcule une donnée réelle δ en constatant le déplacement réel sur la rétine associée au deux angles du neurone vainqueur
  $modification$: mise à jour des poids des neurones suivant une fonction de voisinage hn_μ en apprenant δ
end for

Afin d'explorer l'ensemble de l'espace moteur avec cet apprentissage,les angles moteurs de n sont bruités avant de construite δ. Ce bruit est un bruit blanc (il suit une loi normale) de moyenne 0 et d'écart type variable dans le temps. Cet écart type est en fait une estimation de la moyenne des erreurs constatées à l'apprentissage entre n et δ[1][2].

4.2 Mise en oeuvre Pratique

Dans la pratique, nous avions prévu d'utiliser une webcam, ainsi qu'un bras robotique contrôlé par ordinateur.
Ce bras robotique aurait été muni d'un marqueur spécial (blanc) sur l'extrémité du bras permettant ainsi de la repérer via la webcam.
Nous aurions alors pu tester l'algorithme d'exploration par babutiements rétinien. Via les informations données par la webcam, nous aurions donné les instructions de l'algorithme aux moteurs du bras. Cela n'a pas été possible dû à des problèmes avec le firmware java que nous voulions utiliser.

4.3 Difficultés rencontrées

Au début, nous n'avions aucune connaissances concernant les réseaux de neurones. Un travail d'étude à donc dû être réalisé[3]. De temps en temps, la mauvaise interprétation de la théorie a conduit à des erreurs d'implémentation. Nous avons aussi rencontrés quelques problèmes plus techniques, que ce soit en Java(thread, interface graphique) ou avec le Robot. En effet, nous n'avons pas pu utilisé le firmware java que nous avions prévu d'utiliser afin de contrôler le bras robotique à cause de problèmes techniques.

4.4 Perspectives d'utilisation

Le programme pourra être utilisé dans le but de réaliser des simulations d'un cas pratique ou encore dans un but pédagogique pour la compréhension des réseaux de neurones.
De nombreuses améliorations sont bien sûr envisageables (meilleur gestion de l'affichage, paramétrisation plus poussée. . .).

5 Conclusion

Ce module d'initiation à la recherche nous a permi de comprendre le fonctionnement des réseaux de neurones, comment contrôler un bras robotique avec ces derniers avec différentes méthodes.
Ils nous a également permi d'avoir un aperçu du monde de la recherche et de son fonctionnement.

Bibliographie

[1] N. Rougier et Y.Boniface. *Dynamic self-organizing map.*

[2] N. Rougier et Y.Boniface. *Motivated Self-organizing.*

[3] C. Touzet. *Les réseaux de neurones artificiels : introduction au connexionnisme.* EC2 éd., 1992.

[4] Wikipedia. Carte auto adaptative.