# Cours-TD d'introduction à l'Intelligence Artificielle

## Partie VI

# L'apprentissage non-supervisé

Simon Gay

# Introduction à l'Intelligence Artificielle

- **Menu :**

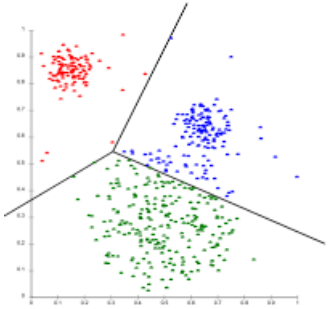
  - Théorie :
    - Les réseaux non-supervisés
    - Méthode d'apprentissage Winner-Take-All
    - Les cartes auto-organisatrices (SOM)
    - Exemple de SOM
    - Les *growing* SOM

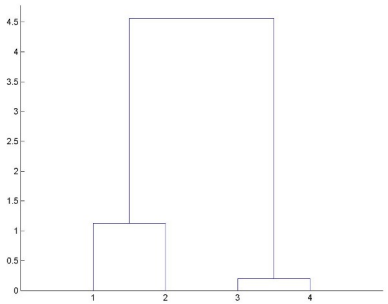
  - Pratique :
    - implémentation d'une carte SOM

# Introduction à l'Intelligence Artificielle

- **Apprentissage non supervisé**
  - Deux principales catégories

    Méthodes de partitionnement  Méthodes de regroupement hiérarchique

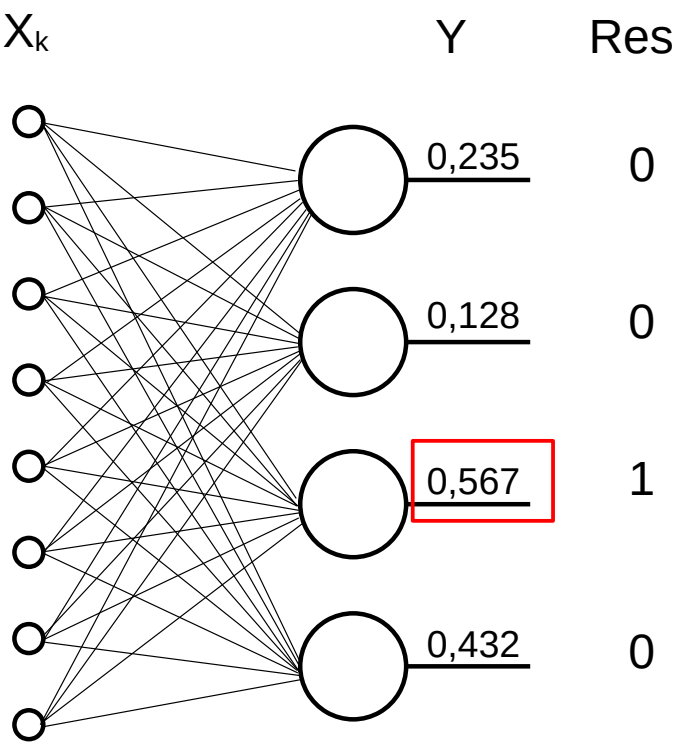





  - Les réseaux de neurones non-supervisés sont principalement des méthodes de partitionnement
    - Mémoires associatives
    - Cartes auto-organisatrices,
    - Réseaux attracteurs
    - ...

# Introduction à l'Intelligence Artificielle

- **Apprentissage non supervisé**
  - Comment apprendre sans résultat connu ?
  - Méthode la plus répandue : le **Winner-Take-All**
  - Chaque neurone voit ses poids initialisés aléatoirement
  - Compétition entre les neurones :
    - Un des neurones aura une sortie supérieure aux autres (neurone gagnant)
    - Le neurone gagnant prend pour résultat 1
    - Les autres neurones sont inhibés (résultat=0)



# Introduction à l'Intelligence Artificielle

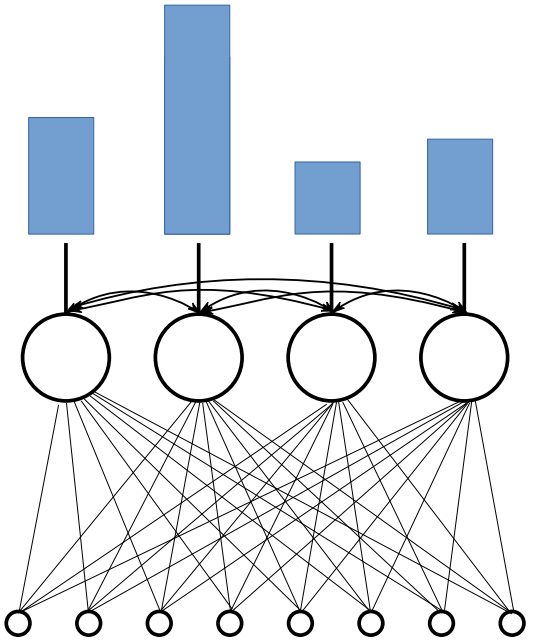
- **Apprentissage non supervisé**

  - Deux approches : les réseaux récursifs et le partitionnement
  - Réseau récursifs :
    - Neurones connectés les uns aux autres
    - Le neurone gagnant inhibe les autres

      les poids sont modifiés jusqu'à stabilisation

      avec le gagnant comme seul actif

  - Applications
    - Sélection d'action (robotique)
    - Vision stéréoscopique



# Introduction à l'Intelligence Artificielle

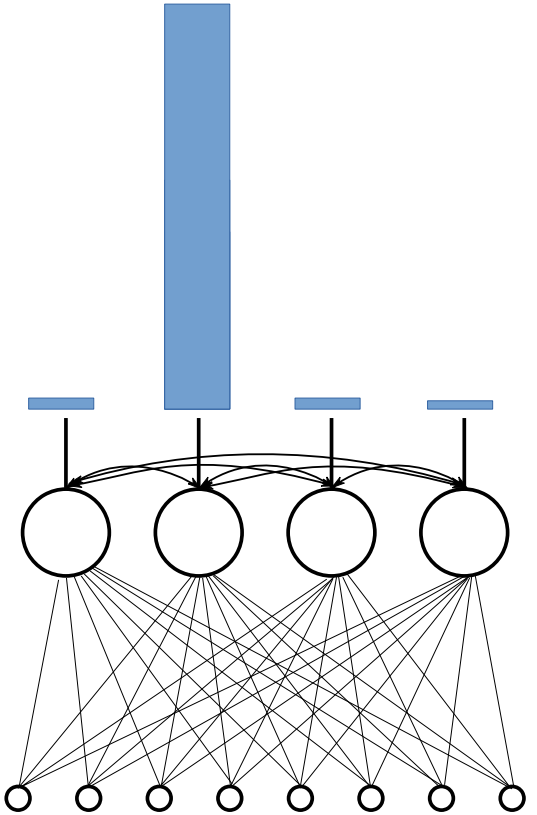
- **Apprentissage non supervisé**

  - Deux approches : les réseaux récursifs et le partitionnement
  - Réseau récursifs :
    - Neurones connectés les uns aux autres
    - Le neurone gagnant inhibe les autres

      les poids sont modifiés jusqu'à stabilisation

      avec le gagnant comme seul actif

  - Applications
    - Sélection d'action (robotique)
    - Vision stéréoscopique



# Introduction à l'Intelligence Artificielle

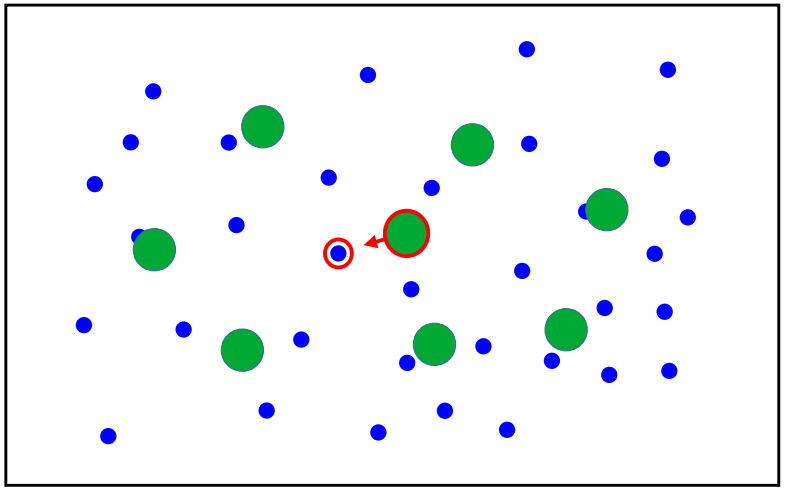
- **Apprentissage non supervisé**
  - Partitionnement des neurones
    - Un nombre N (fixé ou non) de neurones
    - Chaque neurone définit un vecteur W de poids
    - On définit l'activité avec la distance

$$Y_k(X) = \|X - W_k\|$$

  - Le neurone le plus actif est celui dont le vecteur W est le plus proche de X

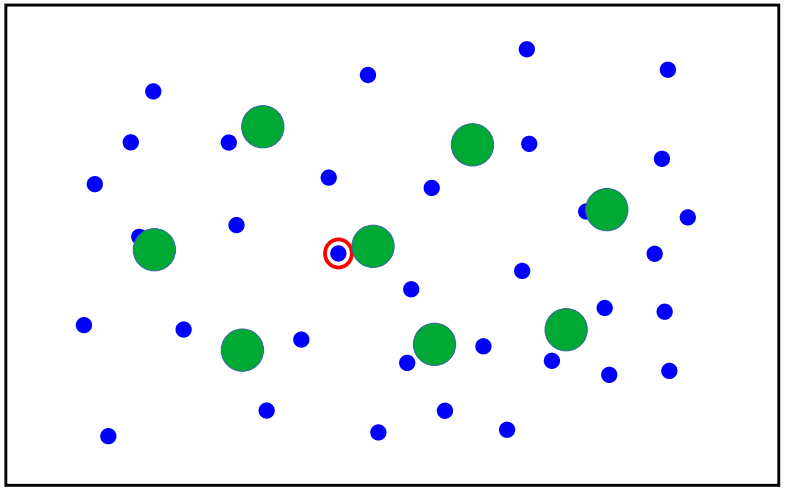
# Introduction à l'Intelligence Artificielle

- **Apprentissage non supervisé**
  - Ce principe implique :
    - Le neurone gagnant se 'rapproche' de l'exemple qu'il a 'gagné'
    - → il se 'spécialise' pour les valeurs proches de cet exemples…
    - … et s'éloigne de toutes les autres



# Introduction à l'Intelligence Artificielle

- **Apprentissage non supervisé**
  - Ce principe implique :
    - Le neurone gagnant se 'rapproche' de l'exemple qu'il a 'gagné'
    - → il se 'spécialise' pour les valeurs proches de cet exemples…
    - … et s'éloigne de toutes les autres



# Introduction à l'Intelligence Artificielle

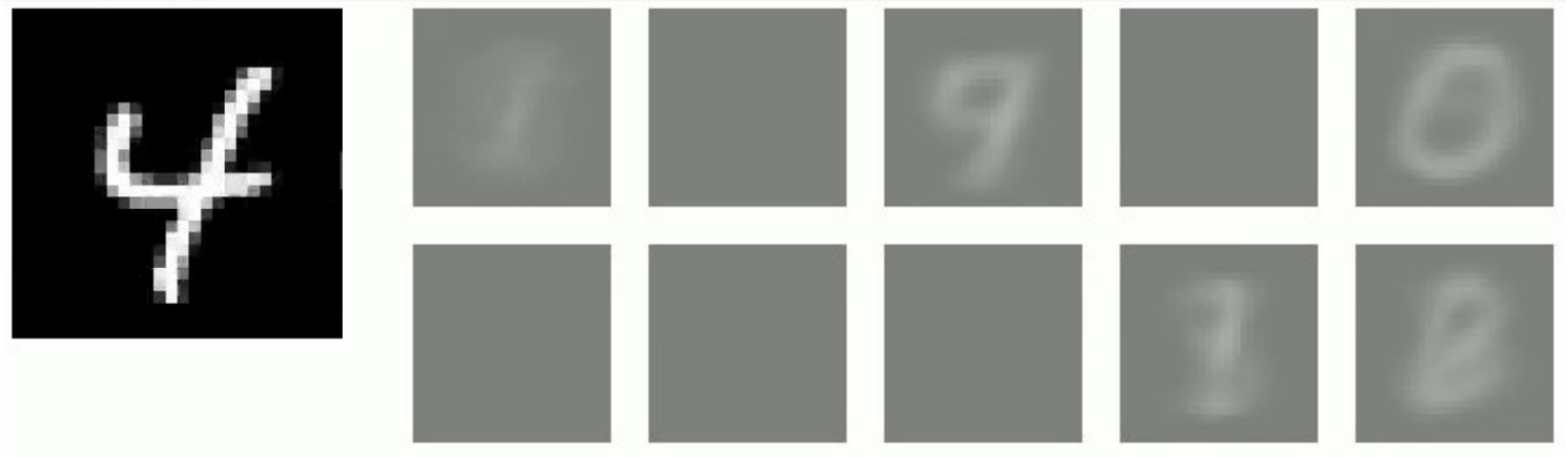
- **Apprentissage non supervisé**
  - Mise à jour des poids du neurone gagnant :

$$w_i^{t+1} = w_i^t + \alpha \times \left( x_i - w_i^t \right)$$

  - On réduit la distance entre le vecteur W du neurone gagnant et X
    - La variation est d'autant plus grande que la distance est importante

# Introduction à l'Intelligence Artificielle

- **Apprentissage non supervisé**
  - Exemples (dataset MNIST) :
    - 10 neurones initialisés aléatoirement

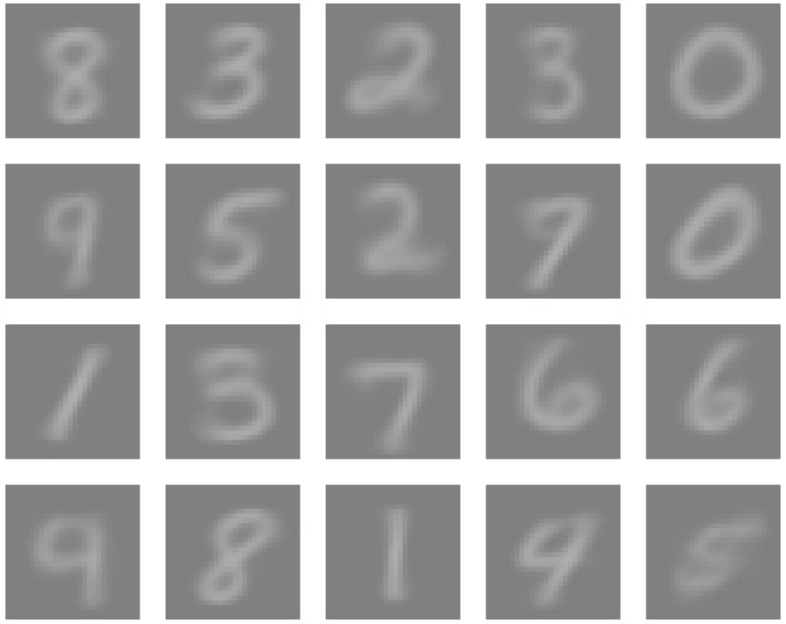


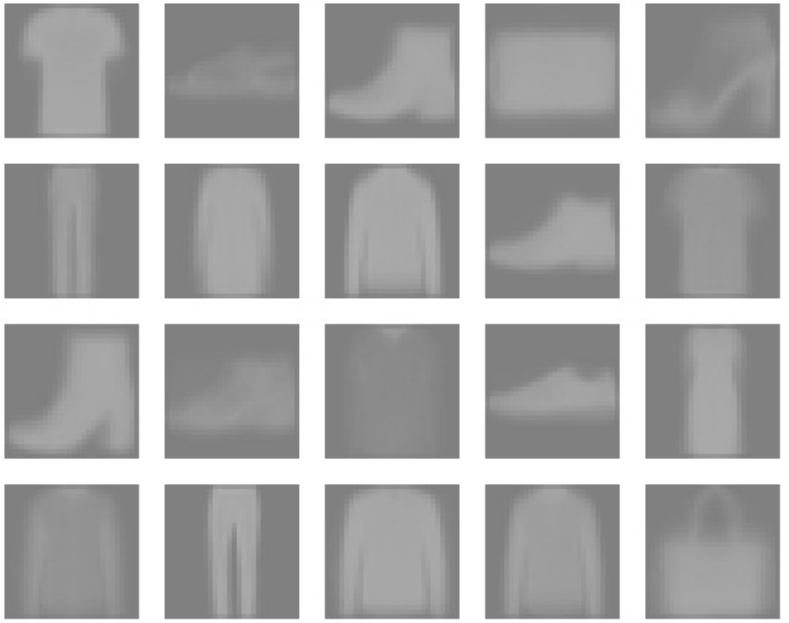
    - Les neurones se spécialisent progressivement sur un type de forme particulier… et laissent le champs libre sur les autres formes pour les autres neurones.

# Introduction à l’Intelligence Artificielle

- Le partitionnement est variable (car poids initialisés aléatoirement)

- Autant de classes que de neurones disponibles





# Introduction à l'Intelligence Artificielle

- **Apprentissage non supervisé**

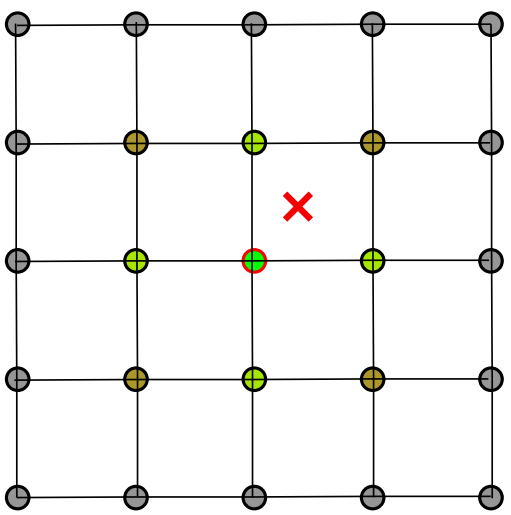
  - Inspiration biologique : les régions corticales encodent les différents stimulis sensoriels et les actions motrices…
  - … mais des points proches dans une région corticale sont associés à des stimuli ou des actions motrices proches
    - Deux points  proches dans le cortex tactile sont associés à deux points proches sur la peau
    - Deux points proches dans le cortex parietal correspondent à deux postures proches.
  - Les régions corticales permettent d'encoder l'ensemble de l'espace continu des stimuli et actions motrices possibles.
    - Comme un cortex forme un espace 2D continu, on peut interpoler entre deux neurones contiguës pour caractériser toute les possibilités sensorimotrices

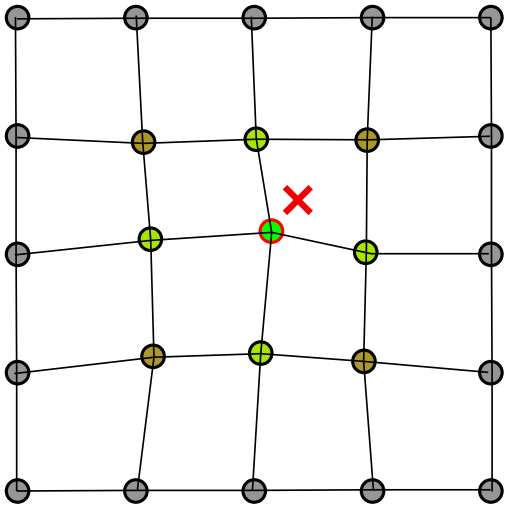
# Introduction à l'Intelligence Artificielle

- **Apprentissage non supervisé**

  - En 1984, le statisticien Teuvo Kohonen propose un principe pour reproduire ces propriétés : la carte auto-adaptative (self-organizing map, SOM)
    - Les neurones sont organisés sous forme d'une matrice 2D
    - Le neurone 'gagnant' partage sa victoire avec les neurones voisins

      → les neurones sont mis à jour avec un facteur qui dépend de la distance par rapport au gagnant





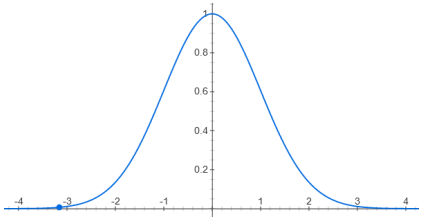
# Introduction à l'Intelligence Artificielle

- **Apprentissage non supervisé**
  - Carte auto-adaptative
    - Une matrice 2D de neurones
    - Principe du Winner-Take-All :
    - Les neurones sont mis à jour d'après la formule suivante :
      - Soient *r* le neurone gagnant et *k* les autres neurones

$$w_i^{t+1} = w_i^t + \alpha \times h(r,k) \times (x_i - w_i^t)$$

    - $\alpha$ est le *coefficient d'apprentissage*, *h(r,k)* est la *fonction de voisinage*
    - La fonction de voisinage caractérise la distance entre un neurone n et le gagnant r.
      - Fonction strictement décroissante sur [0 ;+∞[, comprise dans [0;1]
      - Fonction la plus courante :

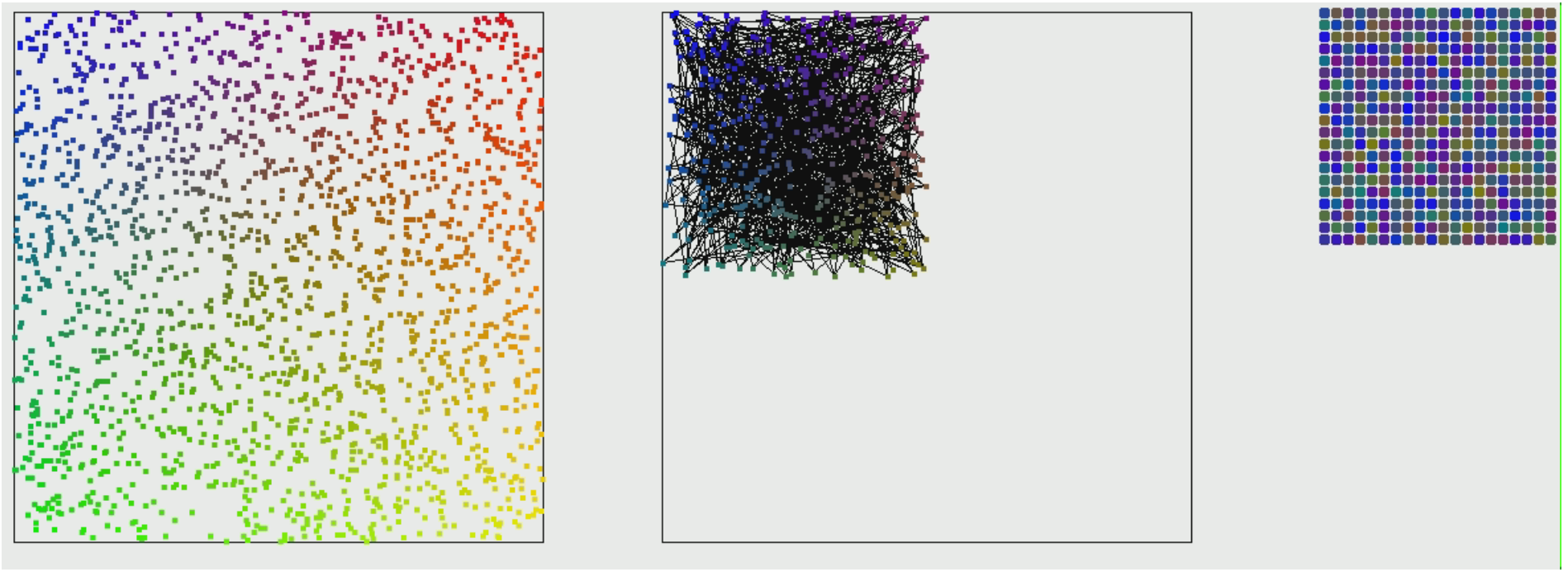
$$h(r,k) = e^{\frac{-d(r,k)^2}{2\times\sigma^2}}$$




    - $\alpha$ et $\sigma$ peuvent varier au cours du temps

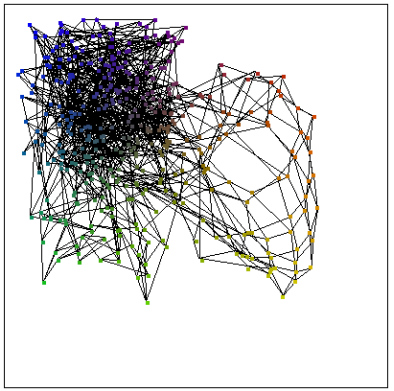
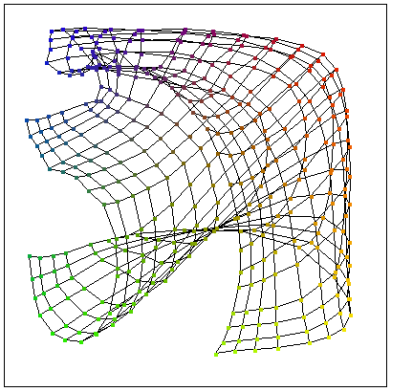
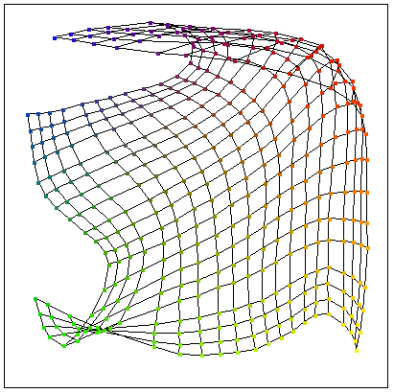
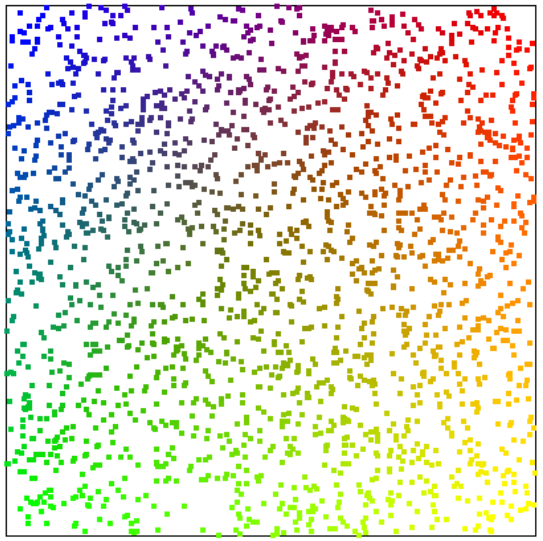
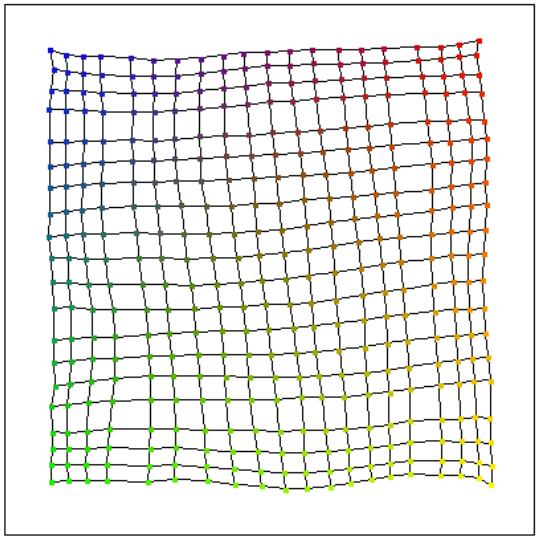
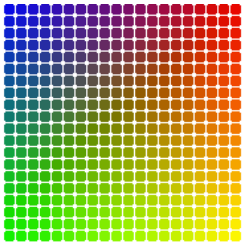
# Introduction à l'Intelligence Artificielle

- **Apprentissage non supervisé**
  - Carte auto-adaptative : exemple sur un espace d'entrées à 2 dimensions
    - Neurones (au centre) initialisés aléatoirement



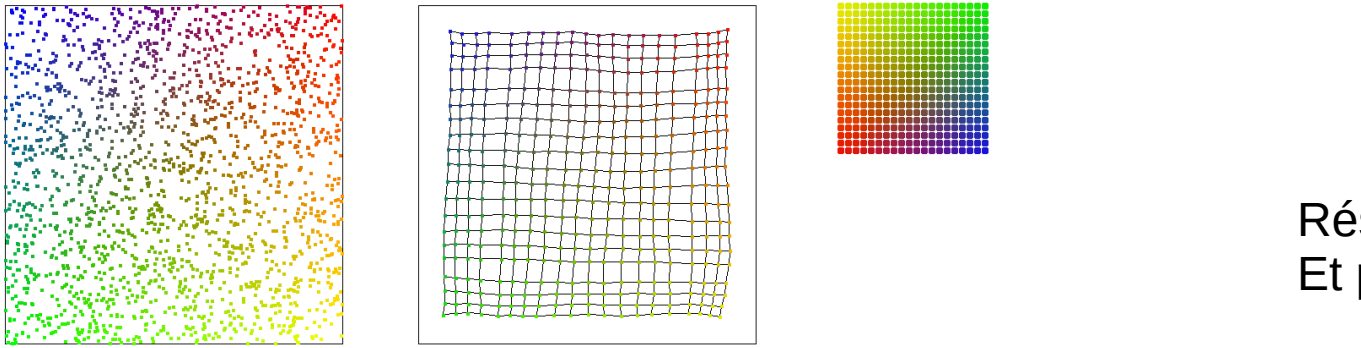
# Introduction à l'Intelligence Artificielle

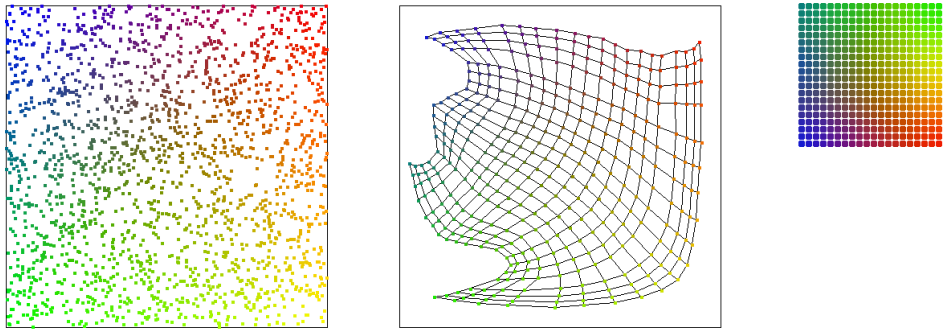
- **Apprentissage non supervisé**
  - Carte auto-adaptative : exemple

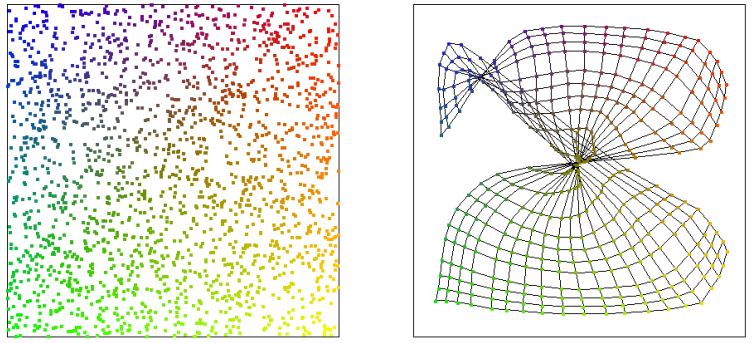







# Introduction à l'Intelligence Artificielle

- **Apprentissage non supervisé**



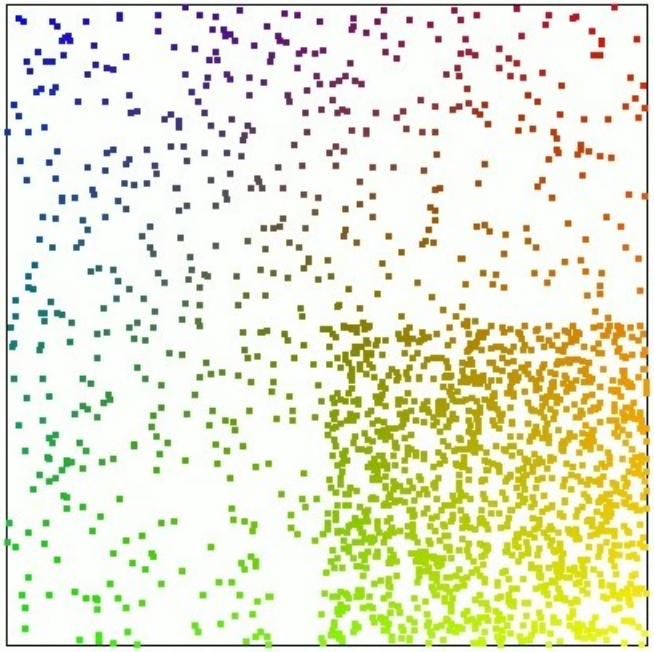


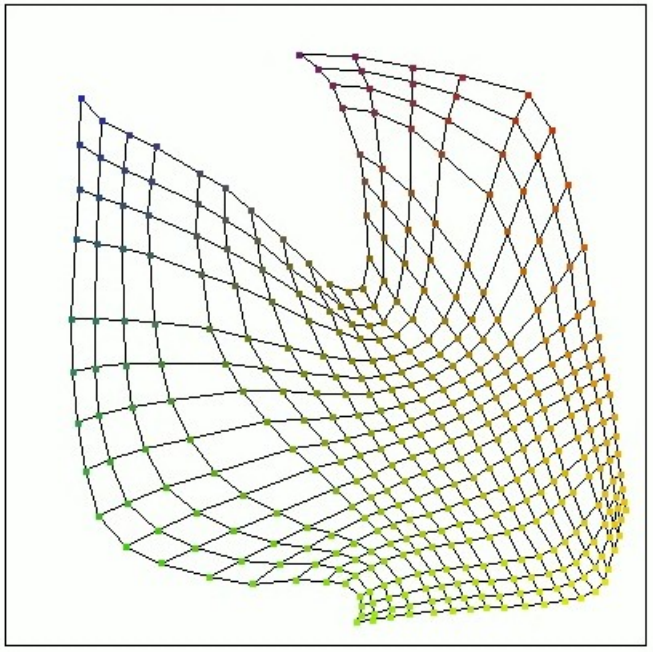


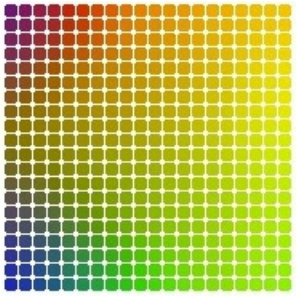
Résultats très variables
Et pas toujours optimal

# Introduction à l'Intelligence Artificielle

- **Apprentissage non supervisé**
  - À noter : la densité des neurones sur une partie de l'espace des entrées dépend du nombre d'échantillons dans cette partie

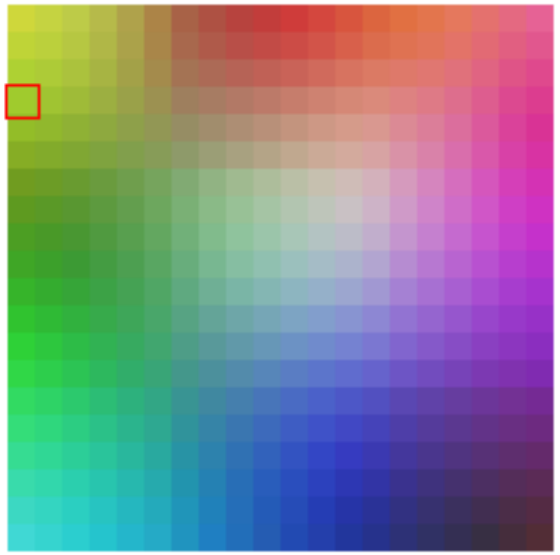


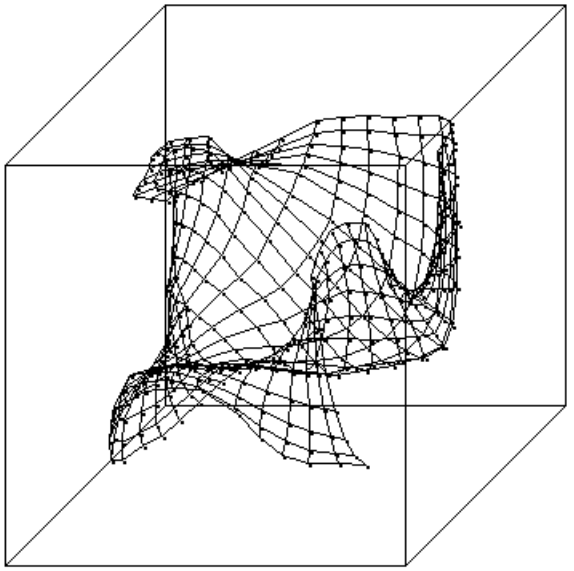




# Introduction à l'Intelligence Artificielle

- **Apprentissage non supervisé**





Passons à la pratique !

# Introduction à l'Intelligence Artificielle

- **L'environnement**
  - Ouvrez Eclipse
  - Créez un nouveau projet Java 'SOM'
  - Créez une classe **Main** et une classe **Neuron**

```
1
2
3  public class Main {
4
5
6      public Main(){
7
8      }
9
10     public static void main(String[] args) {
11
12         new Main();
13     }
14
15 }
16
```

# Introduction à l'Intelligence Artificielle

- **L'environnement**

  - Téléchargez sur ***https://github.com/gaysimon/MachineLearning*** les classes DisplayFrame et DisplayPanel, et intégrez-les au projet
  - Ajoutez dans main une matrice de *Neuron* et un *DisplayFrame*, que vous initialiserez

```
2  public class Main {
3
4      private DisplayFrame display;        // display panel
5
6      public Neuron[][] SOM;
7
8      //////////////
9      public Main(){
10
11         SOM=new Neuron[20][20];
12         for (int i=0;i<20;i++){
13             for (int j=0;j<20;j++){
14                 SOM[i][j]=new Neuron(3);
15             }
16         }
17
18         display=new DisplayFrame(this);
```

# Introduction à l'Intelligence Artificielle

- **Le neurone : plus simple que le neurone formel**
  - Dans la classe *Neuron* :
    - Un vecteur (tableau) de float pour les poids
    - Un float pour le résultat
  - Le constructeur de *Neuron* doit avoir le nombre de poids en paramètre
    - Initialisez le vecteur de poids avec ce nombre
    - Chaque poids doit être initialisé aléatoirement entre 0 et 1
      - Dans Main, vous ajouterez la valeur 1 au constructeur du neurone
  - Créez une fonction **`float compute(float[ ] img)`**
    - Implémentez la fonction qui calcule le résultat du neurone
    - La fonction doit enregistrer le résultat

# Introduction à l'Intelligence Artificielle

- **Le neurone**

```
2  public class Neuron {
3
4      public float[] weights;
5      public float output;
6
7      ////////////////////////////////////////////
8      public Neuron(int size){
9          weights=new float[size];
10         for (int i=0;i<size;i++){
11                 weights[i]=(float)Math.random();
12         }
13     }
14
15     ////////////////////////////////////////////
16     public void compute(float[] img){
17         output=0;
18         for (int i=0;i<weights.length;i++){
19                 output+=(img[i]-weights[i])*(img[i]-weights[i]);
20         }
21         output=(float) Math.sqrt(output);
22     }
23
```

Inutile car on cherche le min

# Introduction à l'Intelligence Artificielle

- **Le neurone**
  - Implémentez la fonction `void learn(float[] img, float ratio)`
    - Le ratio servira de coefficient de voisinage h
    - Ajoutez à la classe une variable learnRate initialisée à 0,02

$$w_i^{t+1} = w_i^t + \alpha \times h(r,k) \times (x_i - w_i^t)$$

# Introduction à l'Intelligence Artificielle

- **Le neurone**

```
2  public class Neuron {
3
4      public float learnRate=0.02f;
```

```
28     ////////////////////////////////////////////
29     public void learn(float[] img, float ratio){
30         for (int i=0;i<weights.length;i++){
31                 weights[i]+= learnRate * ratio * (img[i]-weights[i]);
32         }
33     }
```

# Introduction à l'Intelligence Artificielle

- **Le système d'entraînement**
  - Préparez un vecteur d'entrées input (type float, taille 3) et initialisez-le
  - Préparez une boucle de 100 000 itérations. Elle servira pour entraîner le réseau

```
8     public float[] input;
9
```

```
19    //////////////
20    public Main(){
21
22        // initialize structures
23        input=new float[3];
24
25        SOM=new Neuron[20][20];
26        for (int i=0;i<20;i++){
27            for (int j=0;j<20;j++){
28                SOM[i][j]=new Neuron(3);
29            }
30        }
31
32        // initialize display frame
33        display=new DisplayFrame(this);
34
35        for (int k=0;k<100000;k++){
36
37
38        }
```

# Introduction à l'Intelligence Artificielle

- **Le système d'entraînement**
  - Chaque vecteur d'entraînement correspondra à une couleur aléatoire de l'espace RVB ( Math.random() )

```
for (int k=0;k<100000;k++){

    for (int c=0;c<3;c++) input[c]=(float) Math.random();
```

  - Calculez la valeur de sortie des neurones de la matrice

```
// compute neurons
for (int i=0;i<20;i++){
    for (int j=0;j<20;j++){
        SOM[i][j].compute(input);
    }
}
```

# Introduction à l'Intelligence Artificielle

- **Le système d'entraînement**
  - Il faut déterminer quel neurone est le plus proche de l'exemple présenté.
    - Déterminez les coordonnées nx et ny du neurone avec la valeur minimale

```
12      public int nx=0;
13      public int ny=0;
14      public float min=1;
```

```
54              min=1;
55              for (int i=0;i<20;i++){
56                  for (int j=0;j<20;j++){
57                      if (SOM[i][j].output<min){
58                          min=SOM[i][j].output;
59                          nx=i;
60                          ny=j;
61                      }
62                  }
63              }
64
```

# Introduction à l'Intelligence Artificielle

- **Le système d'entraînement**
  - Nous allons nous occuper de la mise à jour des neurones
  - Créez la fonction de voisinage `float neighbor(float x)`
    - Nous utiliserons en paramètre le carré de la distance

$$h(r,k) = e^{\frac{-d(r,k)^2}{2\times\sigma^2}}$$

    - Utilisez $\sigma = 2$

```
public float neighbor(float x){
    return (float)Math.exp(-x/(8));
}
```

# Introduction à l'Intelligence Artificielle

- **Le système d'entraînement**
  - Mettez à jour les neurones de la matrice avec la fonction

    ```
    public void learn(float[] img, float ratio)
    ```

  - Pour le ratio, on utilise la fonction de voisinage. Calculez $d^2$

```
for (int i=0;i<20;i++){
    for (int j=0;j<20;j++){
        SOM[i][j].learn(input, neighbor( (nx-i)*(nx-i)+(ny-j)*(ny-j)));
    }
}
```

# Introduction à l'Intelligence Artificielle

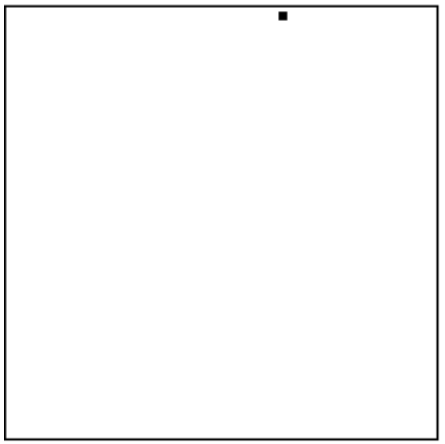
- **Le système d'affichage**
  - Ajoutez dans DisplayPanel.java l'affichage du vecteur d'entrées et la 'couleur' des neurones

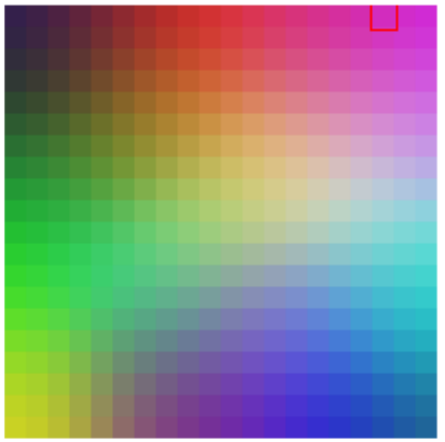
```
g.setColor(Color.black);
g.drawRect(0, 0, 200, 200);
g.fillOval(-2+(int)(main.input[0]*200), -2+(int)(main.input[1]*200), 5, 5);
```

```
float val1,val2,val3;
for (int i=0;i<20;i++){
    for (int j=0;j<20;j++){
        val1=main.SOM[i][j].weights[0];
        val2=main.SOM[i][j].weights[1];
        val3=main.SOM[i][j].weights[2];
        g.setColor(new Color(val1,val2,val3));
        g.fillRect(250+i*10, j*10, 10, 10);
    }
}
```

# Introduction à l'Intelligence Artificielle

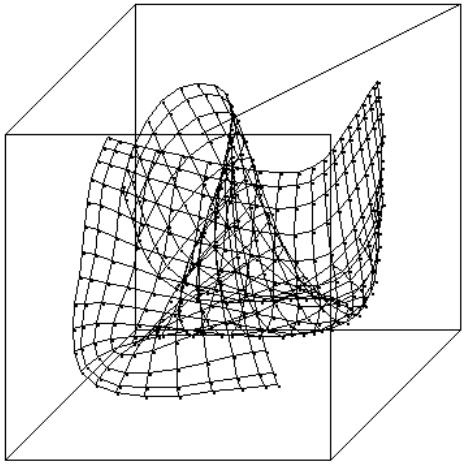
- **Testez votre système !**





  - Les neurones couvrent avec une matrice 2D un espace 3D de couleurs

Vous trouverez sur le github le code pour afficher la répartition des neurones en 3D



# Introduction à l'Intelligence Artificielle

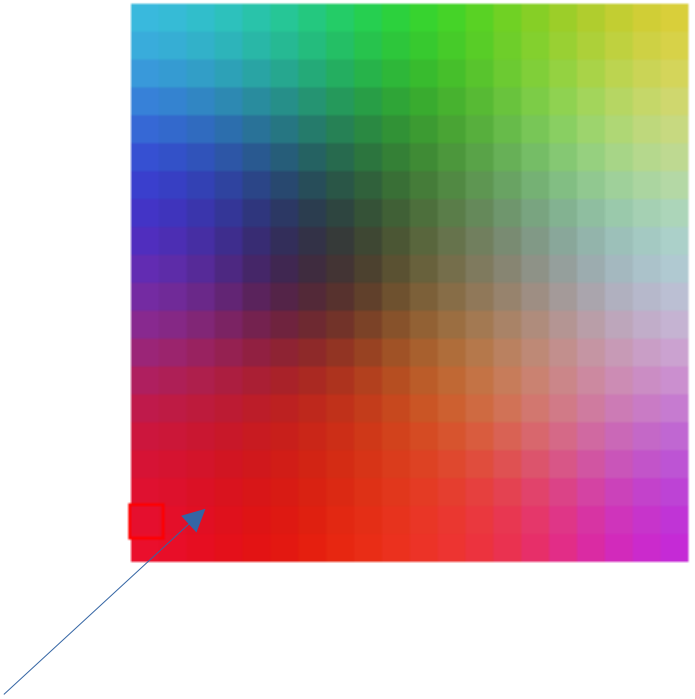
- **Amusons-nous un peu avec l'ensemble des couleurs**
  - Ajoutez le code suivant pour forcer une sur-représentation de certaines couleurs

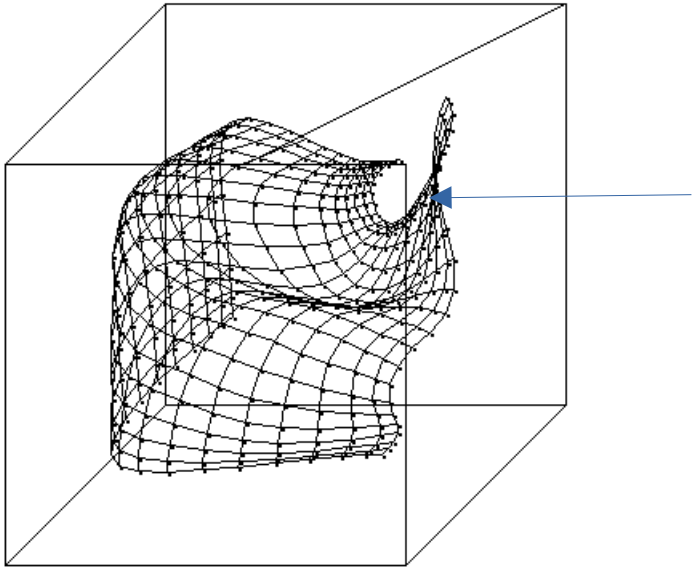
```
for (int c=0;c<3;c++) input[c]=(float) Math.random();

if (Math.random()<0.25){
    input[0]=input[0]/4+0.75f;   // red
    input[1]=input[1]/4;         // green
    input[2]=input[2]/4;         // blue
}
```

  - Vous pouvez tester avec d'autres couleurs

# Introduction à l'Intelligence Artificielle

- **Amusons-nous un peu avec l'ensemble des couleurs**

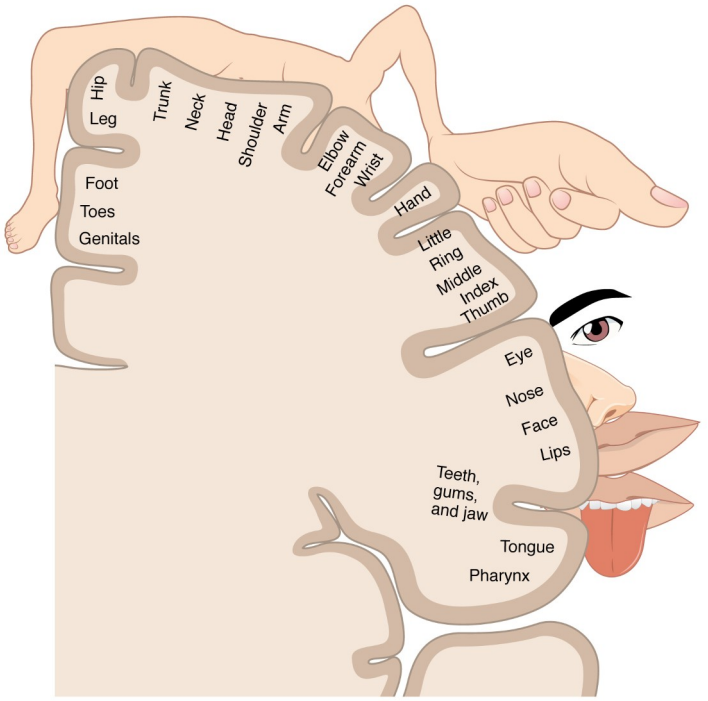




  - Les couleurs sont également sur-représentées sur la carte auto-adaptative

# Introduction à l'Intelligence Artificielle

- **Cette propriété est aussi valable dans le cortex tactile !**

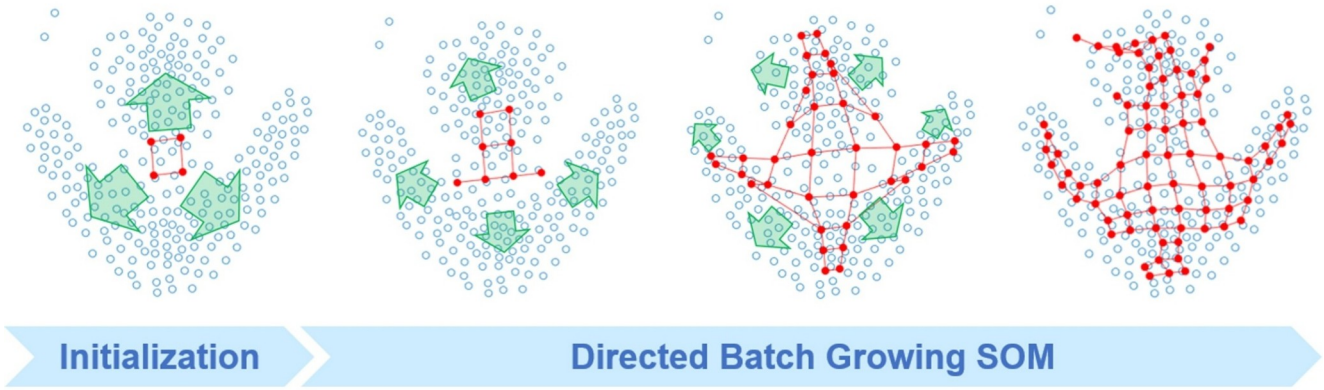



Homonculus de Penfield

- Même observations sur le cortex visuel (fovéa sur-représentée dans le cortex V1)

# Introduction à l'Intelligence Artificielle

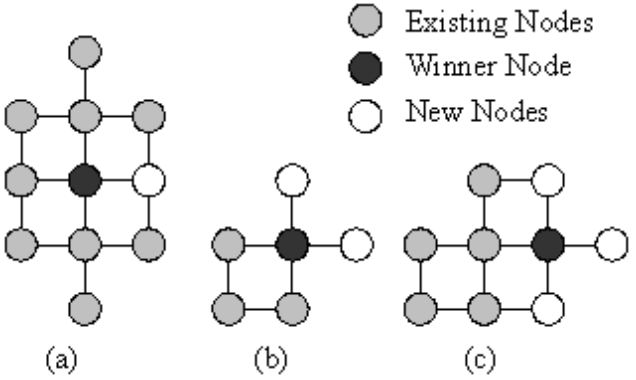
- **Variante du SOM : le Growing Self-Organizing Map**
  - Le nombre de neurone augmente progressivement
  - Le réseau s'étend dans la 'direction' ou se trouvent les données




Source : Mahdi Vasighi & HomaAmini, *A directed batch growing approach to enhance the topology preservation of self-organizing map, Applied Soft Computing, 2017*

# Introduction à l'Intelligence Artificielle

- **Variante du SOM : le Growing Self-Organizing Map**
  - Le réseau commence avec un petit nombre de neurones (en général 4)
  - Si le neurone gagnant est en bordure de réseau, et que la distance est supérieure à un seuil prédéfini, des neurones voisins sont ajoutés au réseau




  - Le réseau va 'grandir' jusqu'à couvrir l'ensemble des valeurs d'entrées