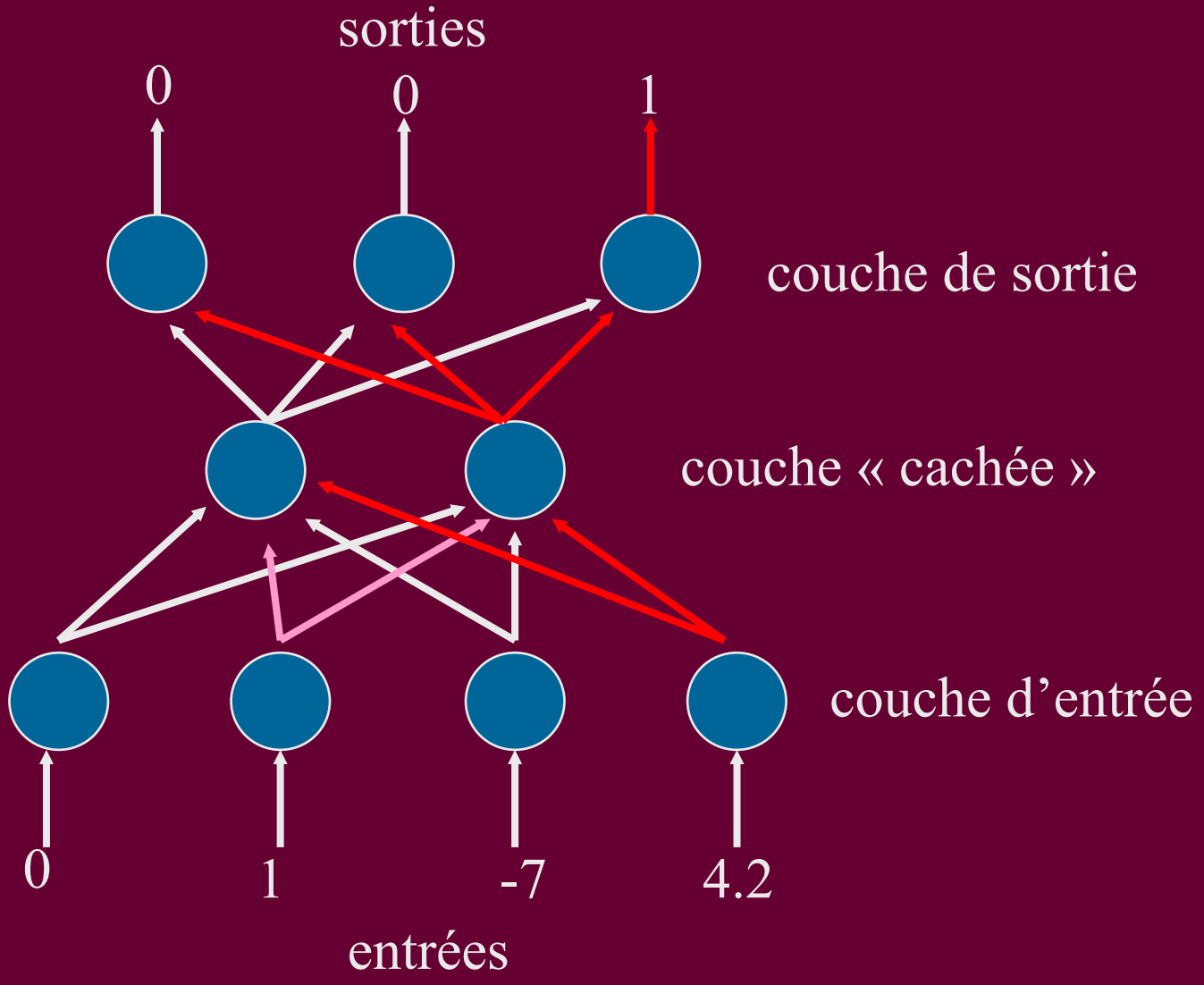
réseaux de neurones
et connexionnisme
résumé de cours et exercices

Pierre De Loor CERV-ENIB deloor@enib.fr

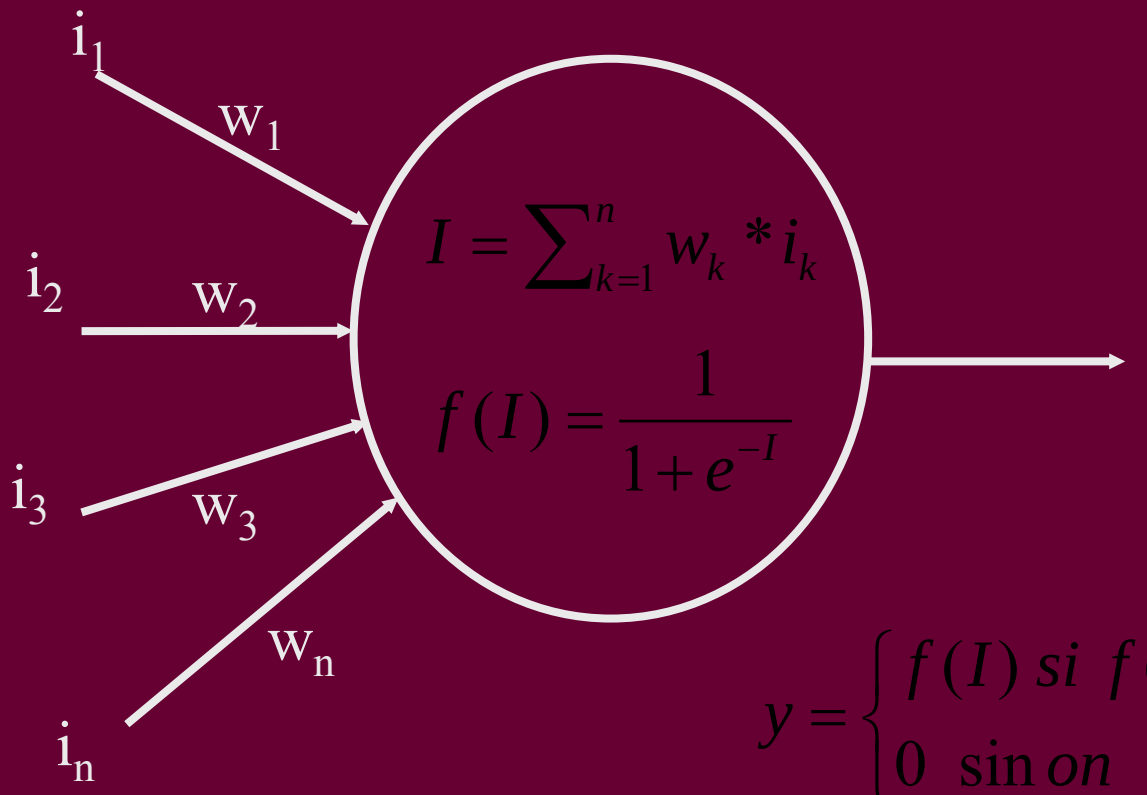
Les objectifs du connexionnisme

- Imitation du cerveau
  - Absence de localisation de l'information
  - Fonctionnement massivement parallèle
  - Créer artificiellement une signification
  - Auto organisation
  - Émergence de configurations globales issues de connexions entre éléments simples
  - Système non déterministe

Un premier réseau de neurone



Un premier neurone formel



$$y = \begin{cases} f(I) \; si \; f(I) > T \\ 0 \quad \sin on \end{cases}$$

Une première illustration avec JavaNNS

- Démarrez javaNNS
- Chargez le réseau font.net
  - 576 neurones d'entrées
  - 2 paquets de 24 neurones cachés
    - 1 pour les lignes
    - 1 pour les colonnes
  - 75 neurones de sorties
  - 150 patterns de 75 caractères (2 par caractères)
  - Au chargement, le réseaux reconnaît ces caractères
  - Vérifiez le
  - Réinitialisez-le
  - Tester l'apprentissage tel qu'il est configuré en observant la progression des erreurs

Le perceptron

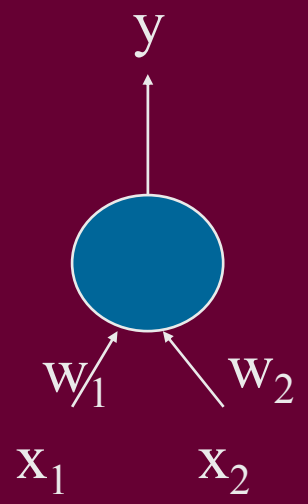
Le Perceptron

Warren Mc Culloh et Walter Pitts 1943

- Vecteur d'entrées continues + ou –
- Sortie +1 ou -1 (selon catégorie)

$$I = \sum_{k=1}^{n} w_k * i_k \qquad y = \begin{pmatrix} +1, si I >= T \\ -1, si I < T \end{pmatrix}$$

$$classiquement \;\; T = 0$$



- Apprentissage (Rossenblatt 1958)
  - Pour chaque nouvel exemple :

$$w_{new} = w_{old} + \beta * y * x$$

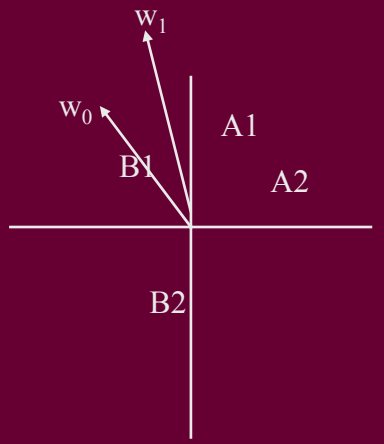
$$\beta = \begin{pmatrix} +1 \; si \; la \; réponse \; est \;\; correcte \\ -1 \, si \, elle \, est \;\; fausse \end{pmatrix}$$

Le Perceptron

exercices

Catégorie A (+1)
A1=(0.3,0.7)
A2=(0.7,0.3)

Catégorie B (-1)
B1=(-0.6, 0.3)
B2=(-0.2, -0.8)



Départ : w_0 =(-0.6,0.8)

A1(0.3,0.7) → I=0.38 → y=1 exemple bon

w_1 =(-0.6+1*1*0.3,0.8+1*1*0.7)

w_1 =(-0.3,1.5)

A2(0.7,0.3)

w_2 =

B1(-0.6,0.3)

w_3 =

B2(-0.2,-0.8)

w_4 =

terminez l'apprentissage
le résultat final re-classe-t-il tous les patterns ?

Le Perceptron

exercices

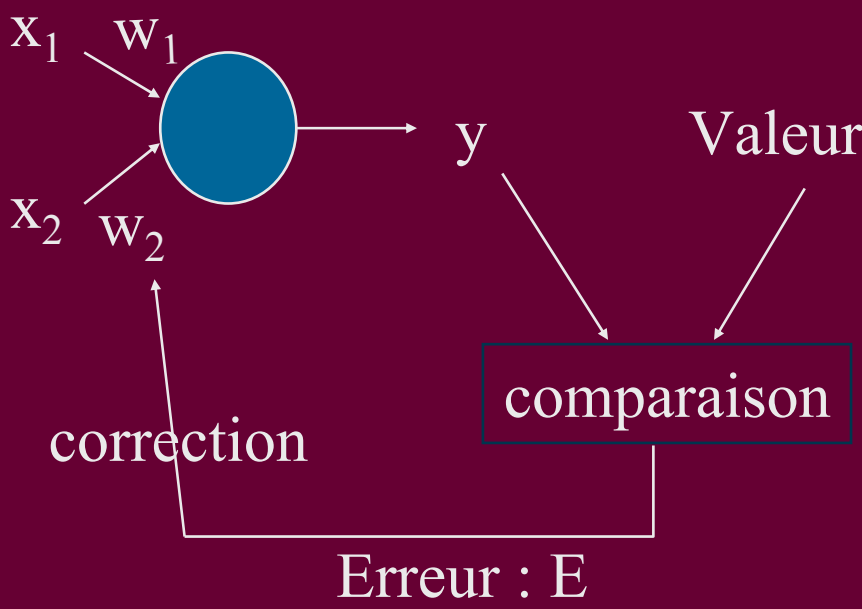
- Tester la classification de nouveaux points. Comment sont-ils classés ? Observer cette valeur relativement au vecteur des poids.
- Implémenter un algorithme de calcul de sortie d'un perceptron (java).
- Implémenter un algorithme d'apprentissage du perceptron.
- Utiliser cet algorithme pour répondre aux questions suivantes :
- Au fur et à mesure que l'on ajoute des points, que se passe-t-il pour le vecteur des poids ?
- Quand le vecteur des poids est à peu près bon, est-ce que quelque chose peut l'amener à diminuer ? Si oui quoi ?
- Quelle conséquence cela a sur l'écriture d'un programme informatique?
- Quelles sont vos remarques concernant l'utilisation du perceptron comme modèle biologique du système neuronal ?
- Que se passe-t-il si un point noté +1 doit désormais être noté -1 ?
- Décrire un problème que le perceptron ne peut pas résoudre.

Apprentissage par minimisation d'erreur

Minimisation d'erreur : *Ada*ptive *line*ar element. Bernard Widrow et Ted Hoff 1960

$$I = \sum_{k=1}^{n} w_k * i_k \qquad y = \begin{pmatrix} +1, siI >= T \\ -1, siI < T \end{pmatrix}$$

$$T = 0$$



$$w_{new} = w_{old} + \frac{\beta * E * x}{|x|^2}$$

$\beta = taux\ d'apprentissage\ (0 < \beta < 1)$

delta-rule

Minimisation d'erreur : *Ada*ptive *line*ar element. Bernard Widrow et Ted Hoff 1960

- La mise à jour des poids n'implique pas une sortie conforme
- Cette mise à jour est 'itérée' autant de fois que nécessaire
- Elle peut modifier la valeur des exemples précédents
- Donc il faut revoir ces exemples

Algorithme d'apprentissage de *Adaline*

```
Pour chaque exemple {
 calculer I, Y et E
 si E\=0 {
  pour l'exemple en cours à tous les exemples précédents {
     calculer E
     si E\=0 {
       calculer L=x1²+x2²+…
       pour chaque poids wi{
         deltai= +(B*E*xi)/L;
         wi=wi+deltai
       }
     calculer E
     }
  }
 }
}
```

- Mécanisme non trivial
- Exemple (avec B=0.5 et w initial = (-0.6;0.8) :
- A1=(0.3,0.7)
- I=-0.18+0.56=0.38
- Y=+1
- Tout va bien
- B1=(-0.6,0.3)
- I=0.36+0.24=0.6
- Y=+1 -> E=-2
- L(B1)=0.36+0.0=0.45
- delta1=(0.5*-2*-0.6)/0.45=1.3
- delta2=(0.5*-2*0.3)/0.45=-0.7

- W=(0.7,0.1)
- Il n'y a plus d'erreur pour B1
  - Car I=-0.39 -> Y=-1
- Il n'y a pas non plus d'erreur pour A1, donc c'est terminé

Adaline
exercices

- Tracer l'évolution du vecteur des poids lors d'un apprentissage d'Adaline sur les exemples suivants :

A = +1	B = -1
A1=(0.3,0.7)	B1=(-0.6,0.3)
A2=(0.4,0.9)	B2=(-0.4,-0.2)
A3=(0.5,0.5)	B3=(0.3,-0.4)
A4=(0.7,0.3)	B4=(-0.2,-0.8)

Taux d'apprentissage B=0.5

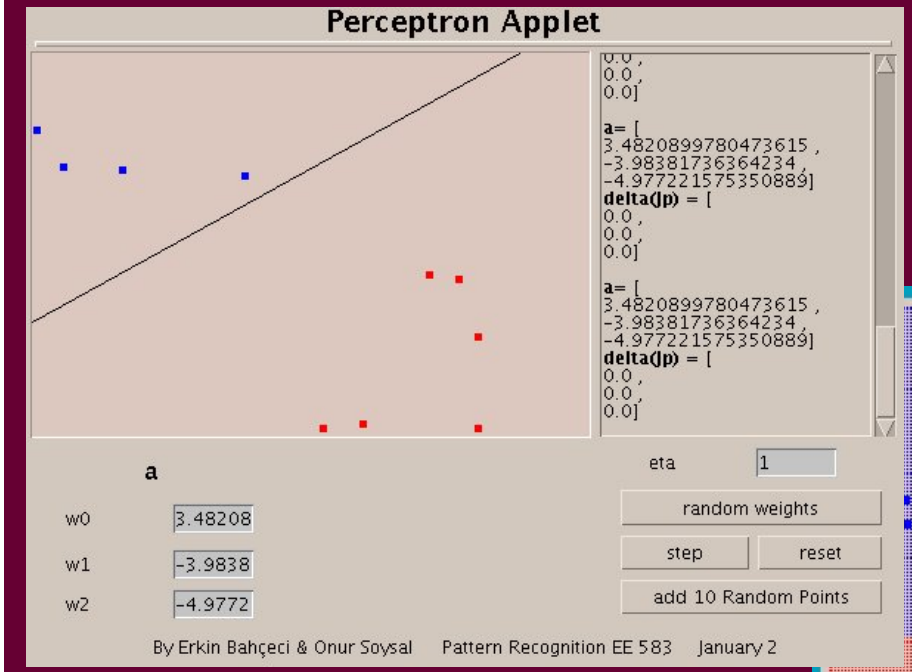
Adaline
exercices

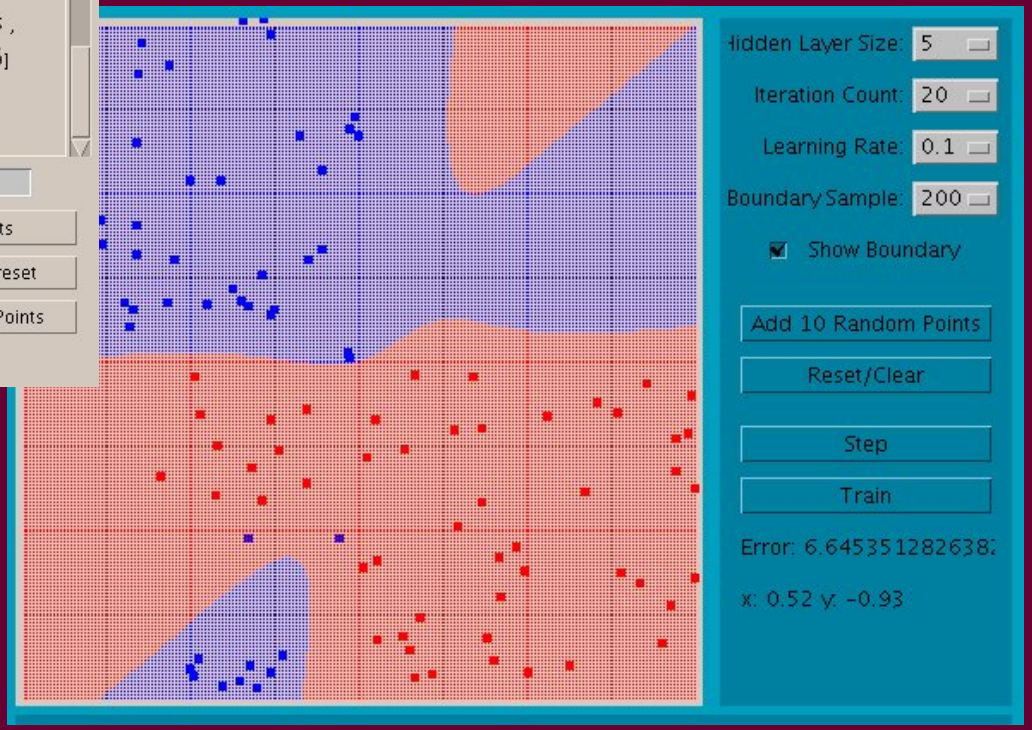
- Combien de fois avez-vous du ajuster les poids pour B1 et au total ?
- Quel est le rôle de la règle delta par rapport aux ‘pattern’ à apprendre
- Supposons que w=(0.8,1.0) et qu’un nouveau pattern B5=(0.6,-0.2) soit à apprendre. Que fait l’algorithme ?
- Combien de fois a-t-il fallu modifier le vecteu
- L’apprentissage est-il terminé ?
- Pourquoi ?
- Tenter de faire apprendre un XOR à Adaline

Un peu d'histoire

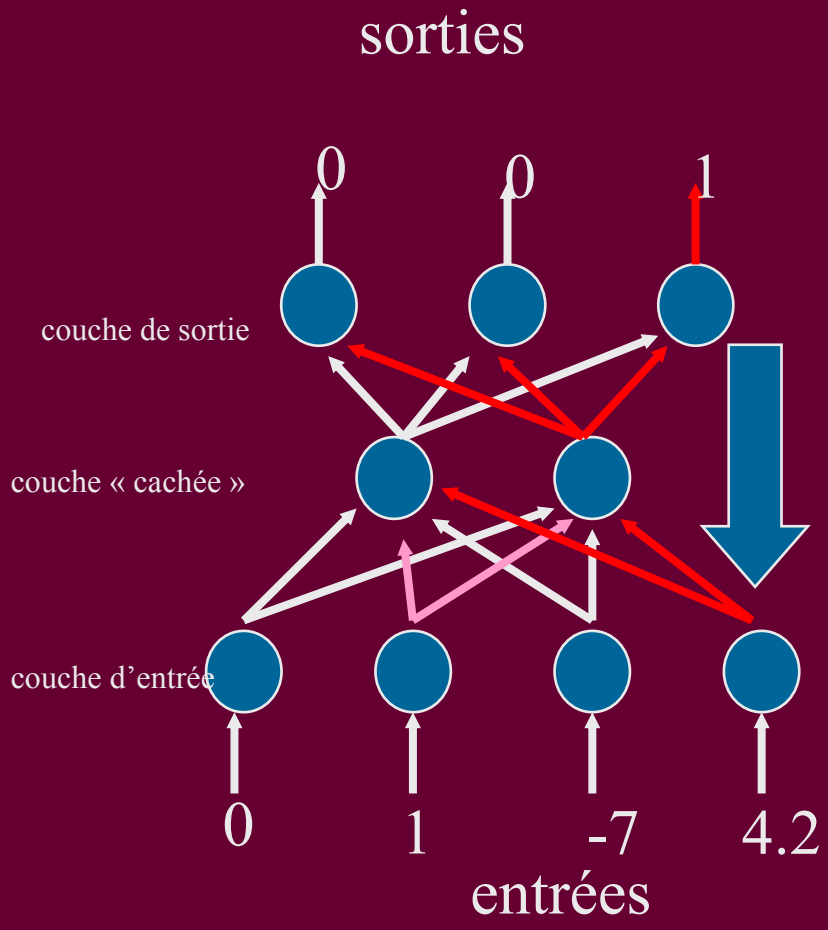
- Minsky et Papert (1969) : Perceptron
  - Montrent que les problèmes non linéairement séparables ne peuvent être appris
  - Donc les RDN ne peuvent pas être utilisés pour ce type de problème (on voulait leur faire faire de la logique = raisonnement)
  - N'imaginent pas qu'il soit possible de mettre en œuvre des procédures d'apprentissage sur plusieurs couches de neurones formels
- Rumelhart et Le Cun (1985) : Retro-Propagation

Du perceptron aux couches cachées





Rétro-propagation



$$I = \sum_{k=1}^{n} w_k * i_k$$

$y = f(I)$ *souvent sigmoide*

$$f(I) = \frac{1}{1 + Exp(-I)}$$

dérivée simple :

$$f'(I) = f(I) - (1 - f(I))$$

$$\Delta w_{ij} = \beta * E * f(I) \quad (0 < \beta < 1)$$

erreur en sortie :

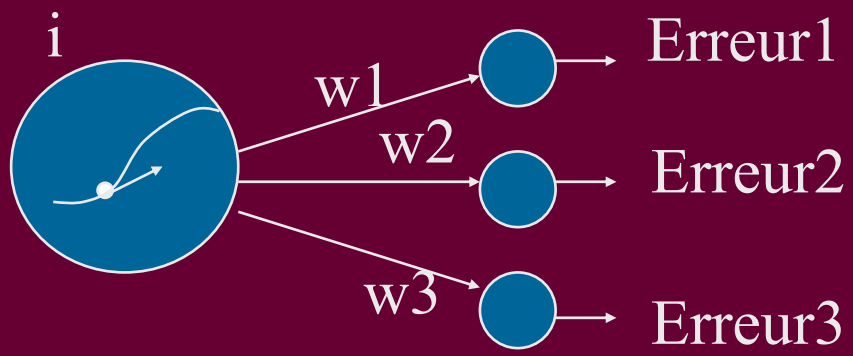
$$E_j^{sortie} = y_j^{desiré} - y_j^{actuel}$$

erreur couche cachée :

$$E_i^{middle} = \frac{d(f(I_i^{middle}))}{dI} \sum_{j=1}^{n} (w_{ij} E_j^{sortie})$$

Retro-propagation

- Pourquoi ça marche ?



$$E_i^{middle} = \frac{d(f(I_i^{middle})}{dI} \sum_{j=1}^{n} (w_{ij} E_j^{sortie})$$

  - Une erreur positive correspond à une sortie trop faible
  - (Réciproquement une erreur négative correspond à une sortie trop forte)
  - Plus il y a d'erreur de sortie, plus i doit être augmenté Pour corriger cette erreur
  - Plus i à d'influence sur une erreur (wj) plus i est facteur d'erreur
  - Plus f(I) à une pente raide, plus il faut la corriger

Retro-propagation

```
Faire {
   Pour chaque pattern {
      Propager l'entrée vers les sorties
      Calculer les erreurs de la couche de sortie
      Calculer les erreurs de la couche cachée
      Ajuster les poids entre c.cachée et c.sortie
      Ajuster les poids entre c.entrée et c.cachée
      }
} Tant que (la somme des erreurs de
  sortie n'est pas acceptable)
```

Test du XOR

- Chargez le XOR dans SNNS
- Tester l'apprentissage avec rétro-propagation et avec souvenirs
- Varier les paramètres des algorithmes et évaluer leur impact sur les temps d'apprentissage

Les neurones ‘unit’ de SNSS

- Attributs importants :
  - Type : input/output/dual/hidden/special
  - activation(a)/bias(θ)/output(o):

$$a_j(t+1) = f_{act}\left(\sum_i w_{ij} * o_i(t), a_j(t), \theta_j\right)$$

$$o_j(t) = f_{out}(a_j(t))$$

  - Exemple : Act_logistic :

$$a_j(t+1) = \frac{1}{1 + e^{-\left(\sum_i w_{ij} o_j(t) - \theta_j\right)}}$$

SNNS : Fonctions d'activations

Fonction	Nom Usuel		Expression
Act_Identity	Linéaire	a(t) =	net(t)
Act_IdentityPlusBias	Linéaire avec Seuil	a(t) =	net(t)+θ
Act_Logistic	Sigmoïde	a(t) =	1/(1+exp(-(net(t)-θ))
Act_Perceptron	Perceptron	a(t) =	0 si net(t) < θ 1 si net(t) > θ
Act_Signum	Binaire	a(t) =	-1 si net(t) < 0 0 si net(t) = 0 1 si net(t) > 0
Act_Signum0	Binaire incluant zéro	a(t) =	0 si net(t) = 0 -1 si net(t) < 0 1 si net(t) > 0
Act_StepFunc	Échelon	a(t) =	1 si net(t) >0 0 si net(t)<=0
Act_Tanh	Tangente Hyperbolique	a(t) = tanh(net(t)+ θ)	

A propos des biais sur NNS

- Le seuil est remplacé par une entrée supplémentaire appelée biais
- Pour le perceptron biais = seuil
- Une représentation analogue est faite pour les autres types de neurones.
- Pour la sigmoide (logistic_function) il y a un paramètre
  - Les unités d'entrées et de sorties n'ont pas de paramètres particuliers, elles transfèrent juste l'information
- Faites des réseaux très simples pour bien comprendre les valeurs des paramètres 'activation, initial activation et bias' de ces unités

SNNS : fonctions de sorties

Fonction	Nom Usuel	Expression
Out_Clip_0_1	Linéaire bornée {0,1}	out(t) = 1 si a(t) > 1 out(t) = a(t) si 0 < a(t) < 1 out(t) = 0 si a(t) < 0
Out_Clip_1_1	Linéaire bornée {-1,1}	out(t) = 1 si a(t) > 1 out(t) = a(t) -1 < a(t) < 1 out(t) = -1 si a(t) < -1
Out_Identity	Linéaire	out(t) = a(t)
Out_Threshold_0.5	Binaire Décalée	out(t) = 1 si a(t) > 0.5 out(t) = 0 si a(t) < 0.5

Retro-propagation avec souvenir (momentum back-propagation)

$$\Delta w_{ij} = \beta * E * f(I) \quad (0 < \beta < 1)$$

Introduction du souvenir

$$\Delta w_{ij} = \beta * E * x_i + \alpha \Delta w_{ij}^{précédent} \quad (0 < \alpha < 1)$$

Ajoute une inertie aux variations des poids
Évite l'accrochage de minimums locaux

précédent : accumulation des variations sur l'**ENSEMBLE** des patterns d'apprentissage !!

Se calcul à partir du passage de tous les patterns

Apprentissage de concepts

oui	non
Moineau	Autruche
Carotte	Persil
Mobylette	Peugeot
Vache	Baleine
Frigidaire	Siemens
	

Apprentissage de concepts

- Énumerez ce à quoi vous font penser ces exemples
- Proposez une dizaine d'attributs pertinents
  - Exemple : naturel/artificiel …
- Testez les avec ces 2 exemples de vérification :
- Faite le apprendre à un réseau de neurones
- Tester l'impact des attributs, du taux d'apprentissage, du type de neurone utilisé, du nombre d'exemples d'apprentissage
- Lien avec les théories de l'apprentissage ?

oui	non
Football	Torball
Avion	Dirigeable
Fusil	Sarbacane

Format des fichiers de pattern SNNS

- Préserver les lignes d'entête comme ceci :

```
SNNS pattern definition file V3.2
generated at 17 Janvier 2006 by Pierre De Loor
# mettre ensuite les commentaires commencent par des #
# ceci est un exemple de pattern pour l'exercice
# les entrées sont 'sait chanter' et 'est petit'
No. of patterns : 3
No. of input units : 2
No. of output units : 1

# Input pattern 1 le moineau :
1   1
# Output pattern 1:
1
# Input pattern 2 l'autruche :
0 0
# Output pattern 2:
0
# Input pattern 2 la carotte :
0 0.25
# Output pattern 2:
1
```

Hebbian Learning

Apprentissage « Hebbien » (Hebb 1949)

- Plus proche des neuro-sciences

Quand un axone de la cellule A est suffisamment proche pour exciter la cellule B de façon répétitive ou persistante, il se produit un processus de croissance ou un changement métabolique tel que l'efficacité de A, en tant que cellule excitant B s'en trouve augmentée.

Des groupes de neurones qui tendent à s'exciter simultanément forment un ensemble de cellules dont l'activité peut persister à la suite du déclenchement d'un événement et peut servir à le représenter

La pensée est l'activation séquentielle de plusieurs groupes d'ensemble de cellules

Apprentissage « Hebbien » (Hebb 1949)

- Stimulus/réponse inconditionnels et conditionnels (Pavlov)
- Aspect dynamique
- Oui mais …
  - Dans quelle proportion (vitesse, borne)?
  - Comment diminuer les liens non significatifs ?
  - Neo-hebbian learning

Neo-hebbian learning

seuil

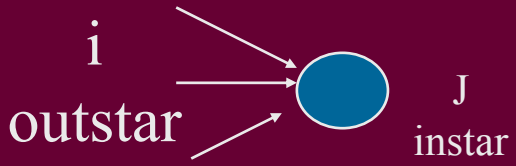
$$y_j(t+1) - y_j(t) = \Delta y_j(t) = -Ay_j(t) + I_j(t) + \sum_{i=1}^{n} w_{ij}(t)[y_i(t-\tau) - T]$$

Vitesse de descente
lorsqu'il ne se passe rien

Stimulus externe

temps
de parcourt

Expression nulle si $y(t-\tau)<T$



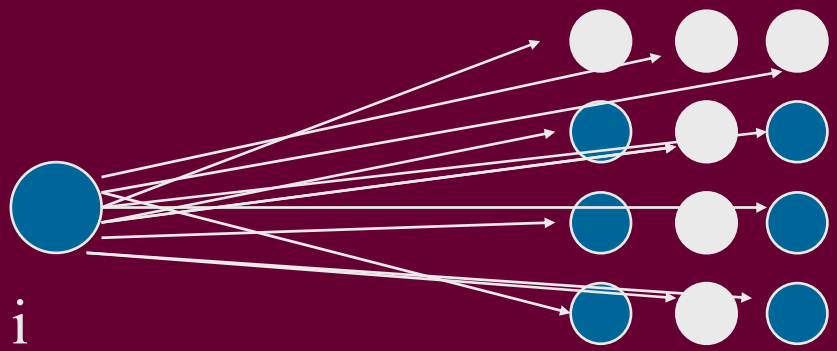
(oubli)

$$w_{ij}(t+1) - w_{ij}(t) = \Delta w_{ij}(t) = -Fw_{ij}(t) + Gy_j(t)[y_i(t-\tau) - T]$$

Loi de Hebb : renforcement si activation simultanée

Exemple

- Apprendre le T



i
outstar

j
instar

Stimulus
externe

Stimulus inconditionné
Réponse inconditionnée
Stimulus conditionné
Réponse conditionnée

Apprentissage Hebbien différentiel

- Pallier aux difficultés suivantes :
  - Se rapprocher du modèle biologique
  - Limiter la valeur des poids
  - Apprentissage négatif (diminution explicite des poids quand deux neurones sont en opposition)

$$\Delta w_{ij} = \beta * \Delta_{yi} * \Delta_{yj}$$

Drive-Reinforcement Theory

- Se rapprocher de l'apprentissage naturel
  - Apprentissage non linéaire
    - Lent au début et rapide ensuite par accumulation d'expérience
  - Mémorisation des stimulus
    - Le stimulus conditionnel avant l'inconditionnel favorise l'apprentissage

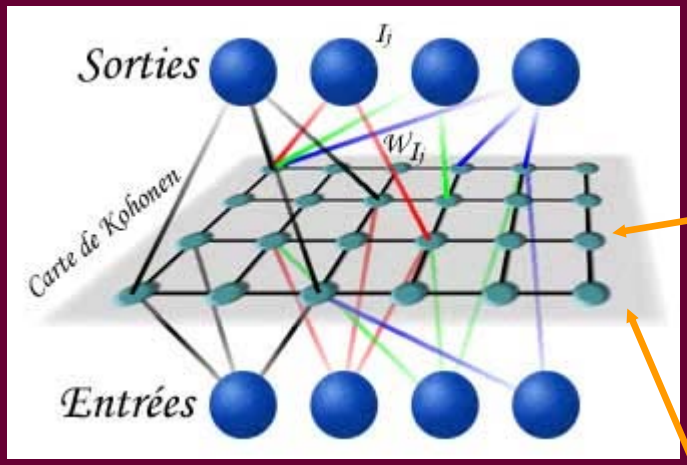
$$\Delta w_{ij}(t) = \Delta y_j(t) \sum_{k=1}^{\tau} \beta \left| w_{ij}(t-k) \right| \Delta y_i(t-k)$$

mémorisation

Apprentissage compétitif

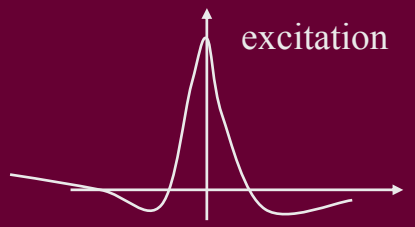
Apprentissage compétitif

- Pas de feed-back : systèmes auto-organisés
- Exemple : Kohonen (inhibition latérale)



Liens d'organisation
Renforce les voisins
Inhibe les neurones éloignés

carte
Auto-organisatrice

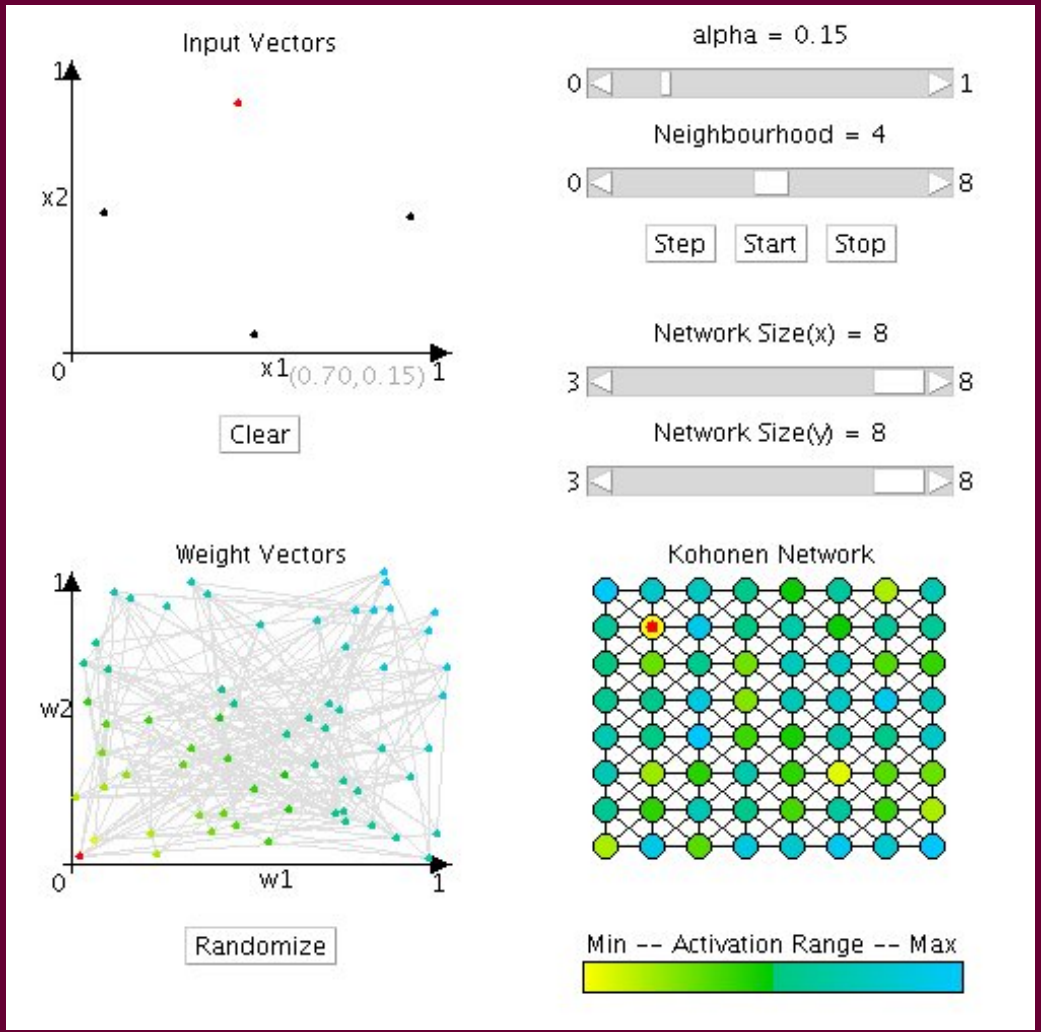


Distance
du neurone gagant

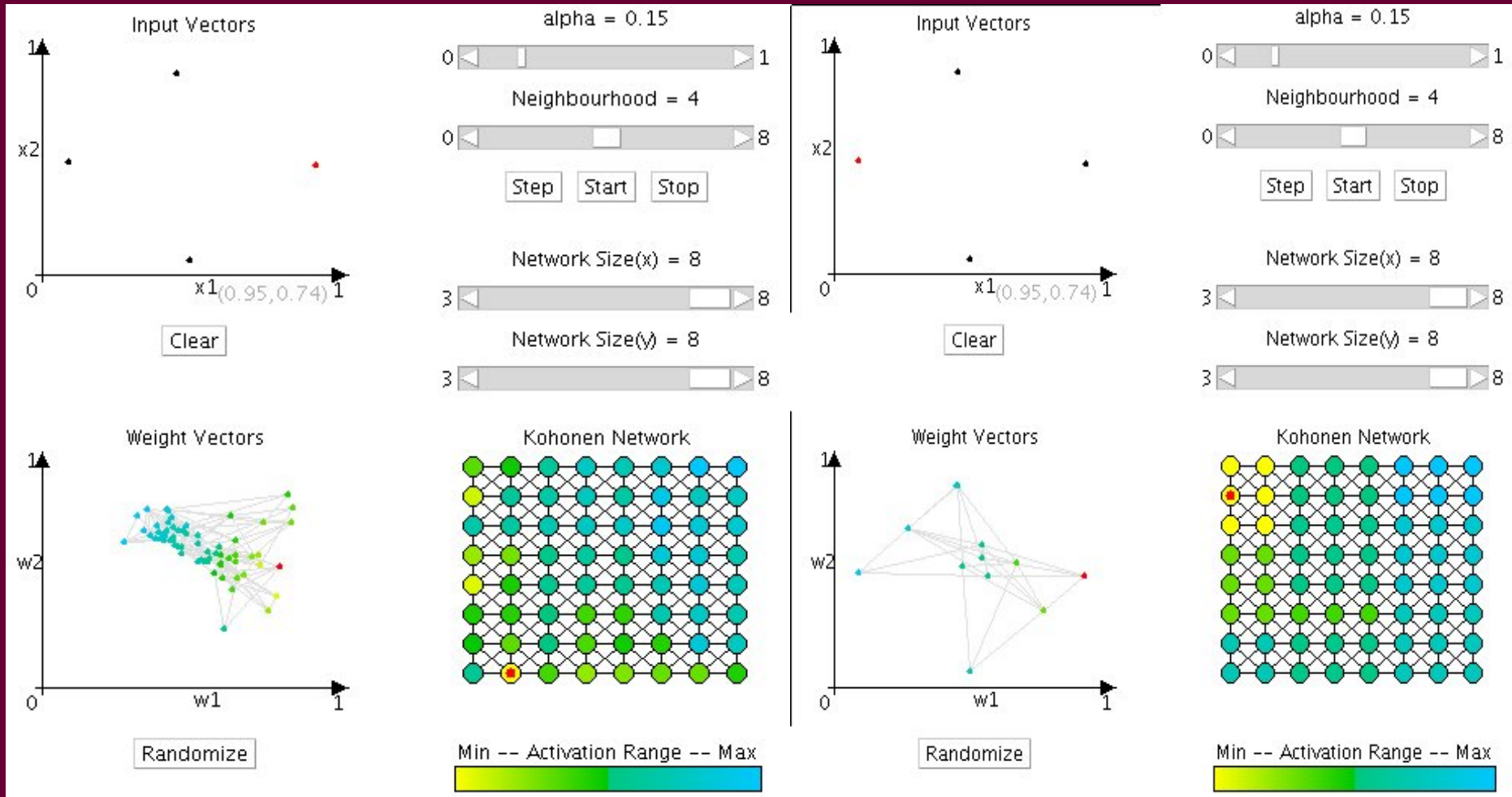
Kohonen

- Vision et ouïe du chat
- Topologie : les neurones activés par les hautes fréquences sont localisés à l'opposé des neurones sensibles au basses fréquences
- Un neurone va représenter une classe
  - Identification de relief
  - Identification de couleurs
  - Deux neurones proches représentent des informations similaires

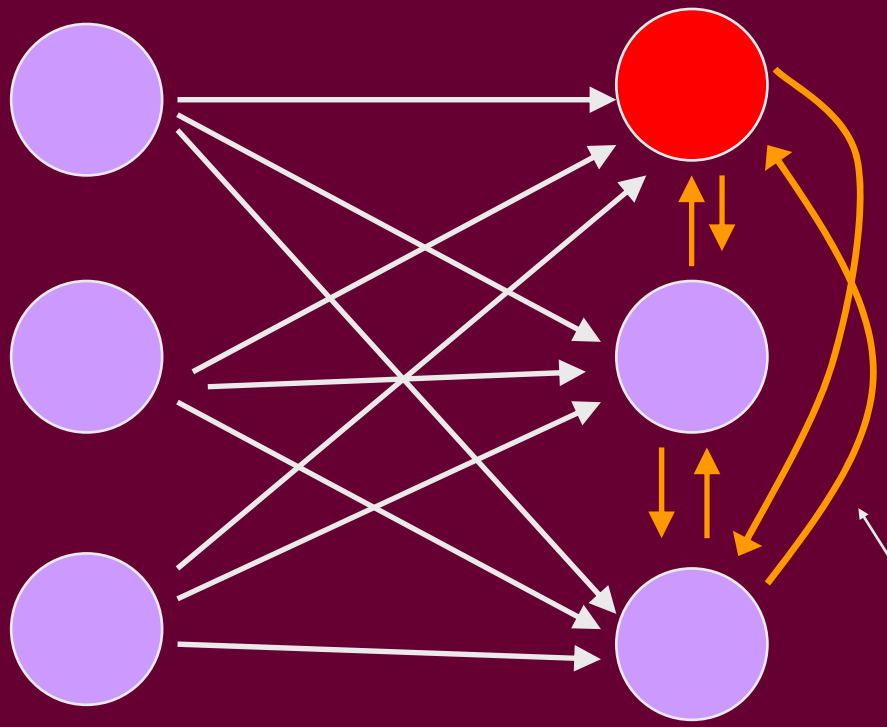
Carte topologique ?



Carte topologique



Election d'un gagnant



Pour une entrée
$I = [I_1 \ldots I_M]$

Un neurone est plus
Adapté que les autres

Les poids Entreé-Kohonen
sont modifiés pour accroitre
Cette adaptation

entrée

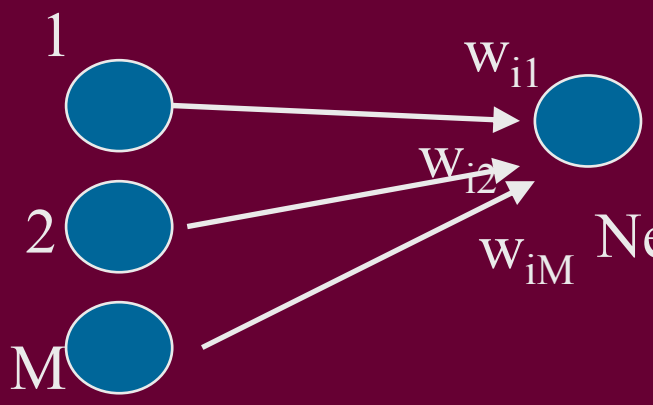
Couche de Kohonen

Effet compétitif implicite
(pas de poids à calculer)

Quel est le neurone le mieux adapté

- Les poids entre couches entrée-Kohonen reflètent une distance entre le neurone et l'entrée

$$di = \sum_{j=1}^{M} (I_j - w_{i,j})^2$$



Si les poids valent les entrées la distance est nulle

Apprentissage

- Selection du neurone gagnant par calcul des distances
- Modification des poids fonction de la distance du gagnant
- La carte s'organise pour
- Si k est le neurone gagnant

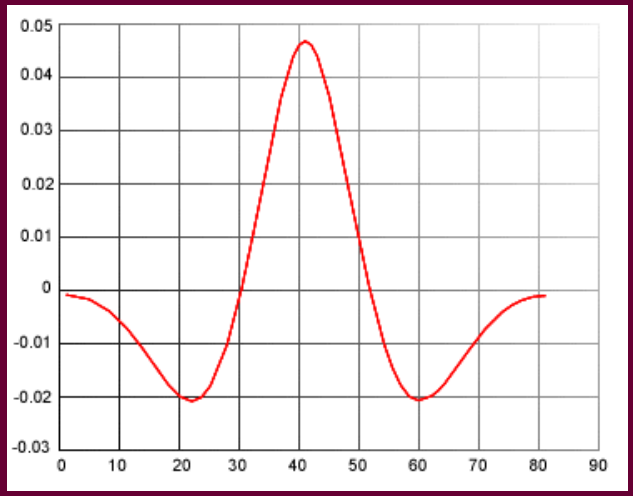
$$w_{kj}(t+1) = w_{kj}(t) + \eta(t) * (I_J(t) - w_{kj}(t))$$

Coefficient d'apprentissage décroissant avec le temps

Apprentissage

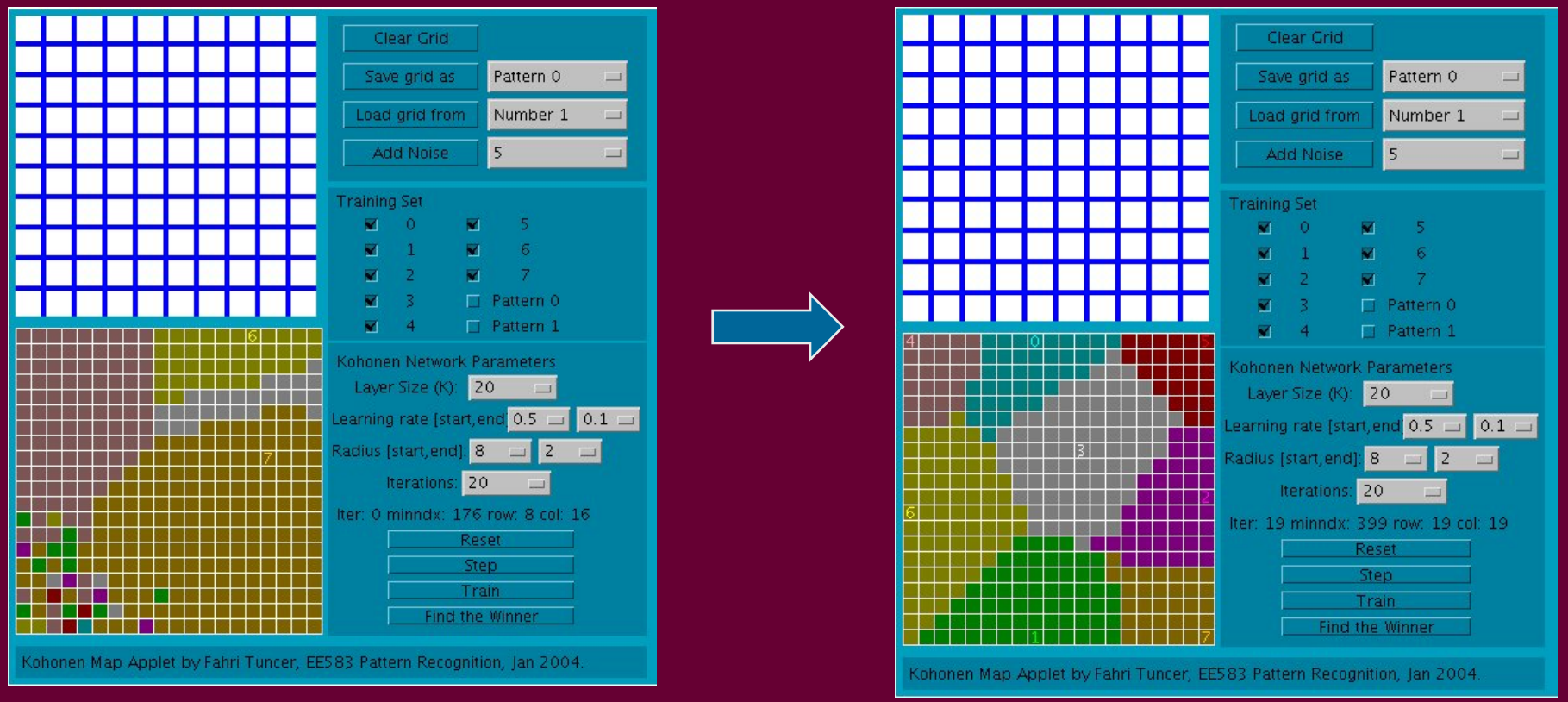
- Pour les autres neurones

$$w_{ij}(t+1) = w_{ij}(t) + \eta(t) * f(dis\tan ce\ de\ k) * (I_J(t) - w_{ij}(t))$$



- Se rétrécie avec le temps

Kohonen autre exemple



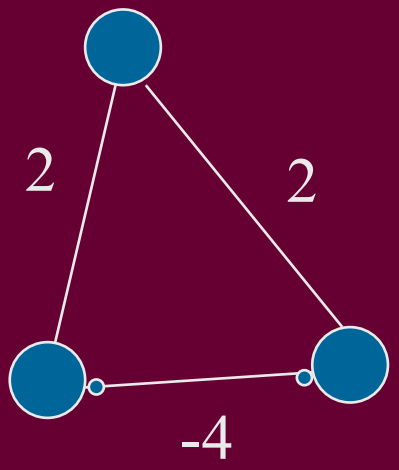
Réseaux attracteurs

Réseaux récurrents

- Il existe des boucles d'influence entre neurones
- C'est plus proche d'un système complexe
- Il est donc plus compliqué de mettre en œuvre un apprentissage

Réseaux de Hopfield ou réseau à attracteur mono couche

- Hopfield



- Activation d'un neurone

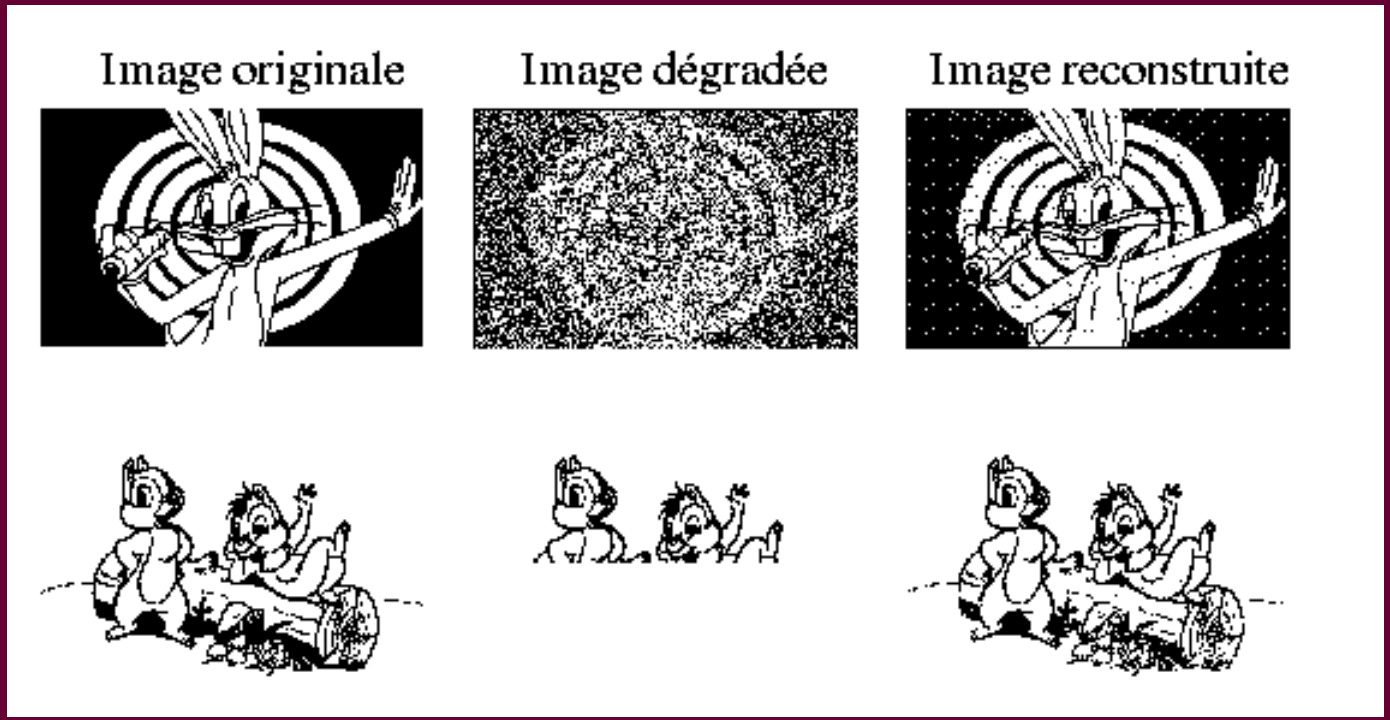
$$I = \sum_{k=1}^{n} w_k * i_k$$

$$y = f(I) = I \; si \;\; I_{\min} < I < I_{\max}$$
$$y = I_{\min} \; si \; I < I_{\min}$$
$$y = I_{\max} \;\; si \; I > I_{\max}$$

- Modèle simplifié :

$$y = 0 \; si \; I \leq 0$$
$$y = -1 \; si \; I > 0$$

Mémoire auto-associative



Apprentissage

- Loi de Hebb
  - Sur p exemples

$$w_{ij} = \sum_{k=1}^{p} x_i^k * x_j^k$$

  - Sur un modèle simplifié, w_{ij} est le nombre de fois où les bits sont égaux – le nombre de fois où ils sont différents
  - Sur modèle simplifié à n neurones on peut apprendre 0.15*n prototypes

Apprentissage des poids en C

```
void CalculateWeights(NET* Net)
{
  INT i,j,n;
  INT Weight;

  for (i=0; i<Net->Units; i++) {
    for (j=0; j<Net->Units; j++) {
      Weight = 0;
      if (i!=j) {
        for (n=0; n<NUM_DATA; n++) {
          Weight += Input[n][i] * Input[n][j];
        }
      }
      Net->Weight[i][j] = Weight;
    }
  }
}
```

- Grille de `i` x `j` neurones
- `NUM_DATA` exemples stockés dans le tableau `Input`
- `Net->Units` : nb neurones

Auto-association par propagation aléatoire

- Les sorties des neurones sont initialisées par les valeurs du pattern à reconnaitre
- Tant qu'il n'y a pas de stabilisation
- Recalculer une sortie choisie aléatoirement

```
void PropagateNet(NET* Net)
{
  INT Iteration, IterationOfLastChange;

  Iteration = 0;
  IterationOfLastChange = 0;
  do {
    Iteration++;
    if (PropagateUnit(Net,
RandomEqualINT(0, Net->Units-1)))
      IterationOfLastChange = Iteration;
  } while (Iteration-IterationOfLastChange
< 10*Net->Units);
}
```

- La sortie de `i` est la somme des sorties des autres neurones `j` liés à `i` et pondérés par le poids `Net->Weight[i][j]`
- Il y a ensuite un seuillage

```
BOOL PropagateUnit(NET* Net, INT i)
{
  INT  j;
  INT  Sum, Out;
  BOOL Changed;
  Changed = FALSE;
  Sum = 0;
  for (j=0; j<Net->Units; j++) {
    Sum += Net->Weight[i][j] * Net->Output[j];
  }
  if (Sum != Net->Threshold[i]) {
    if (Sum < Net->Threshold[i]) Out = LO;
    if (Sum > Net->Threshold[i]) Out = HI;
    if (Out != Net->Output[i]) {
      Changed = TRUE;
      Net->Output[i] = Out;
    }
  }
  return Changed;
}
```

Test d'un réseau de Hopfield

- Utilisez
  - soit l'applet fournie (fermée),
  - soit le code C (ouvert)
- Expérimenter l'apprentissage de différents patterns et éprouvez l'auto-association
- Tester pour différentes tailles de réseaux le nombre de patterns qui sont appris par le réseau
- L'ordre d'apprentissage correspond-il à l'ordre d'oubli ?
- Comment faire pour qu'un réseau apprenne de façon incrémentale ?

Les travaux de Belson et Daucé