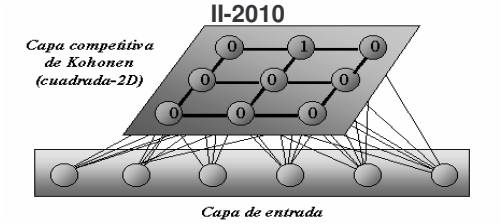
SOM/KOHONEN Network
Mapas Auto-organizativos

Capitulo 5
Redes auto-organizativas
II-2010



Prof. Héctor Allende

Self Organizing Maps

- La red SOM fue creada por Teuvo Kohonen en la década de los 80, rápidamente paso a ser una de las redes **no supervisadas** más importantes que existen hasta el momento.
- SOM es una poderosa herramienta para el trabajo de Data Mining, pero también es utilizada en diversas aplicaciones tales como: reconocimiento de la voz, reconocimiento de textos manuscritos, problemas de optimización, análisis de textura en imágenes y organización de documentos, entre otros.
- Uno de los aspectos más importantes de la red SOM es la preservación topológica de los datos proyectándolos en una malla de dimensión menor.

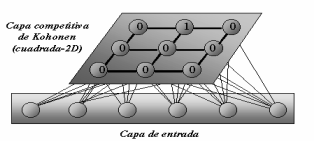
Estructura de la Red

SOM (Self-Organization Map o Kohonen Network)
Es una Red neuronal de aprendizaje no supervisado.

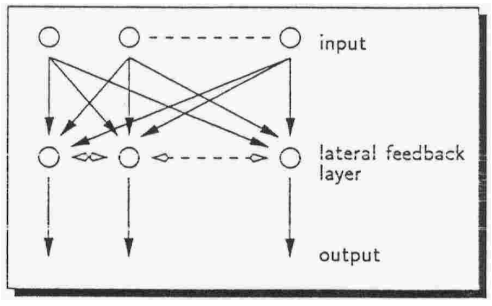
- Posee feedback lateral. En general es de forma indirecta (tipo “Sombrero Mejicano”).
- Posee una única capa, la capa de salida.
- Consiste en un arreglo ordenado de nodos.
- Puede ser unidimensional (K) o multidimensional (KxK)
- La capa de entrada solo distribuye la entrada en la capa de salida (no hay procesamiento). Consiste en N neuronas (dimensión de la entrada).

Self Organizing Maps

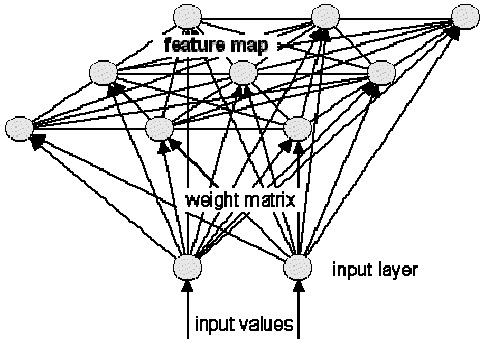
- En la arquitectura de una red SOM, la capa de entrada se conforma por N neuronas (una neurona para cada dimensión del dato) la cual recibe el vector de entrada para pasarlo a las neuronas de la capa de salida.
- La capa de salida esta conformada por K neuronas conectadas de cierta manera la cual forma una malla. En esta capa se produce el procesamiento de la red.



Estructura de la red de Kohonen



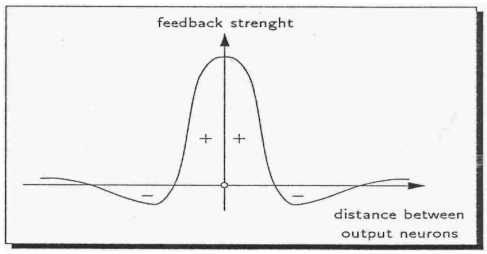
Estructura de la red de Kohonen



Estructura de la Red

- La distancia entre neuronas es (+) para las neuronas más cercanas y (-) para neuronas a distancia media (cero para las restantes).
- La función de feedback influye en la velocidad de aprendizaje, determinando el monto de actualización de las ponderaciones de las neuronas vecinas.
- Vecindad Neuronal: Área afectada por el feedback lateral. (Disminuye con el tiempo)
- Para grandes vecindades, la distancia se puede considerar como una función continua.

Sombrero mejicano



Las Neuronas cercanas reciben un feedback (+)

Las Neuronas a mediana distancia reciben feedback (-)

Las Neuronas lejanas no son afectadas.

Proceso de aprendizaje

- Todas las neuronas incluidas en la vecindad neuronal incluida ella misma participan en el proceso de aprendizaje. Las restantes neuronas no son afectadas.
- El proceso de aprendizaje consiste en cambiar el vector de pesos en la dirección del vector de entrada (feedback positivo).
- Existe también un proceso de olvido; proceso que retarda el progreso (feedback negativo)

Proceso de aprendizaje

- Matriz de pesos: $W = \left[w_{ij} \right]_{\substack{j=1,..,K \\ i=1,..,N}}$
- Vector de entrada: $\bar{x} = \{ x_i \}_{i=1,..,N}$
- Entrada es una función parametrizada $\bar{x} = x(t)$
- Entrada total: neto = W·x
- Si la neurona k que tiene un peso asociado $W(k,:) \in \Re^N$ tal que:

$$\| W(k,:)^T - \bar{x} \| = \min_{j=1,..,K} \| W(j,:)^T - \bar{x} \|$$

entonces se denominada neurona ganadora

Self Organizing Maps, bmu

- Para encontrar la neurona que mejor se asemeja el patrón de entrada presentado (neurona ganadora o *best matching unit*) se calcula la diferencia existente entre el vector de entrada y cada una de las neuronas, para ello es necesario definir una medida de distancia: euclidiana, absoluta, Manhattan, Voronoi, etc.
- Comunmente la distancia utilizada es la euclidiana ya que es fácil de implementar y rápida de calcular, además si los vectores se normalizan a uno es equivalente a maximizar el producto escalar de ambos ($\underline{w}_i^T \cdot \underline{x}$), pero lamentablemente se puede perder información relevante al hacer este proceso.

Sea el vector de entrada $\bar{x} = [x_1, x_2 .... x_n]^T \in \Re^n$

Sea el de pesos $\underline{w}_i = [w_1, w_2 .... w_n]^T \in \Re^n$

$$\underline{w}_{BMU} = \min_i \{ d(\underline{x}, \underline{w}_i) \} \quad i = 1...K$$

Initialization

One old ineffective principle is random initialization of the model vectors.

Random initialization was originally used to show that there exists a strong self-organizing tendency in the SOM, so that the order can even emerge when starting from a completely unordered state, but this need not be demonstrated every time.

On the contrary, if the initial values for the model vectors are selected as a regular array of vectorial values that lie on the subspace spanned by the eigenvectors corresponding to the two largest principal components of input data, computation of the SOM can be made orders of magnitude faster, since:

(i) the SOM is then already approximately organized in the beginning,

(ii) one can start with a narrower neighborhood function and smaller learning-rate factor.

Dinámica de la Red

- **Función de ejecución de la red**:

  Para cada vector de entrada X , $W(j,:)^T \in \Re^n$ la neurona k para la cual

$$\| W(k,:)^T - X \| = \min_{j=1,..,K} \| W(j,:)^T - X \|$$

  se declara ganadora. El ganador es usado para decidir que pesos serán cambiados. Todas las neuronas pertenecientes a la vecindad neuronal participan en el aprendizaje.

Proceso de aprendizaje Lineal

- Aprendizaje lineal: cambios ocurren en dirección de la combinación lineal de x y W(j; :) para cada neurona:

$$\frac{dW}{dt} = \phi(x, W) - \gamma(x, W)$$

donde ϕ y γ son funciones escalares (no lineales).

ϕ : feedback positivo γ : feedback negativo

- A continuación se considera que la vecindad neuronal es toda la red.

Tipos de aprendizaje

- **La ecuación diferencial Trivial:**

$$\frac{dW(j,:)}{dt} = \alpha x^T - \beta W(j,:) \quad \alpha > 0, \beta > 0$$

forma matricial :

$$\frac{dW}{dt} = \alpha \hat{1} x^T - \beta W$$

Condición inicial : $W(0) = W_0$.

Solución :

$$W(t) = \left[\alpha \hat{1} \left(\int_0^t x^T(t') e^{\beta t'} dt' \right) + W_0 \right] e^{-\beta t}$$

Para $t \to \infty$, W(j,:) tiende al promedio exponencialmente ponderado de X y no logra un efecto importante.

Tipos de Aprendizaje

- **La ecuación simple:**

$$\frac{dW(j,:)}{dt} = \alpha a_j(t) x^T - \beta W(j,:) \quad \alpha > 0, \beta > 0$$

forma matricial, $a = Wx$:

$$\frac{dW}{dt} = \alpha 1 a(t) x^T - \beta W = W(\alpha x x^T - \beta I)$$

aprox. en tiempo discreto :

$$\frac{dW}{dt} \to \frac{\Delta W}{\Delta t} = \frac{W(t+1) - W(t)}{(t+1) - t} = W(t)[\alpha x(t) x^T(t) - \beta I]$$

$$\Rightarrow W(t+1) = W(t)[\alpha x(t) x^T(t) - \beta I + I]$$

Condición inicial : $W(0) = W_0$.

Otros Tipos de Aprendizaje

- **La Solución de la ecuación simple**:

$$W(t) = W_0 \prod_{t'=0}^{t-1} [\alpha x(t) x^T(t) - \beta I + I]$$

  – La solución puede ser divergente o convergente a cero, ambos casos son inaceptables.

  – Para tiempos cortos la solución se aproxima a procesos asintóticamente estables.

- Para t ; α relativamente pequeños y $\beta \approx 0$:

$$W(t) \cong W_0 \left[I + \alpha \sum_{t'=0}^{t-1} x(t') x^T(t') \right]$$

Otros Tipos de Aprendizaje

- **La Ecuación diferencial de Riccati**:

$$\frac{dW(j,:)}{dt} = \alpha x^T - \beta a_j W(j,:) \qquad \alpha > 0, \beta > 0$$

como $a_j = W(j,:)x = x^T W(j,:)^T$

$$\Rightarrow \frac{dW(j,:)}{dt} = x^T [\alpha I - \beta W(j,:)^T W(j,:)]$$

En notación matricial :

$$\frac{dW}{dt} = \alpha \hat{1} x^T - \beta (W x \hat{1}^T) \otimes W$$

Otros Tipos de Aprendizaje

- **Ecuación de Riccati:**
  - Considerando un acercamiento estadístico a la ecuación de Riccati, existe una solución, la solución de W es de la forma:

$$\lim_{t\to\infty} W = \sqrt{\frac{\alpha}{\beta}}\hat{1}\frac{< x >^T}{\|< x >\|} \qquad \text{si} < x > \neq \hat{0}$$
$$\text{donde } < x >= E\{x/W\} = cte$$

  - Todo W (j,:) llega a estar paralelo a <x> y tendrá la norma $\|W(j,:)\| = \sqrt{\alpha/\beta}$

Tipos de Aprendizaje

- **Ecuaciones más generales**:

Teorema: Sea $\alpha > 0$, a =Wx y γ(a) una función arbitraria tal que E{γ(a)|W} existe. Sea x = x(t) un vector con propiedades estadísticas estacionarias (e independiente de W). Entonces, si el proceso de aprendizaje es del tipo:

$$\frac{dW(j,:)}{dt} = \alpha x^T - \gamma(a_j)W(j,:) \qquad j = 1,..,\ K$$
$$\text{en notación matricial :}$$
$$\frac{dW}{dt} = \alpha \hat{1} x^T - [\gamma(a)\hat{1}^T] \otimes W$$

tiene soluciones W acotada para $t\to\infty$, entonces debe tener la forma: $\lim_{t\to\infty} W \propto \hat{1} < x >^T$

donde <x> representa la esperanza de x(t).

ie., W(j,:) llega a ser paralelo a <x>

Otros Tipos de Aprendizaje

Sea $\alpha > 0$, a = Wx y γ(a) una función arbitraria tal que E{ γ (a)|W} existe. Sea $<xx^T> = E\{xx^T|W\}$. Sea $\lambda_{max} = máx_i \lambda_i$ el valor propio máximo de $< xx^T >$ y u_{max} el vector propio asociado. Entonces, si el proceso de aprendizaje es del tipo:

$$\frac{dW(j,:)}{dt} = \alpha a_j x^T - \gamma(a)W(j,:)$$
$$\text{en notación matricial :}$$
$$\frac{dW}{dt} = \alpha a x^T - [\gamma(a)\hat{1}^T] \otimes W$$

tiene soluciones no triviales W acotada para $t\to\infty$, entonces debe tener la forma: $\lim_{t\to\infty} W \propto \hat{1} < x >^T$

donde W $u_{max} \neq \hat{O}$, W(0) = W_0 ; ie, W(j,:) llega a ser paralelo a u_{max}

Algoritmo SOM

- La neurona "ganadora" aprende del input. También aprenden las neuronas que topográficamente están cercanas en una distancia geométrica

$$w_i(t+1) = w_i(t) + h_{ci}(t)[x(t) - w_i(t)]$$

Por lo general la función de kernel es Gaussiana

$$h_{ci}(t) = \alpha(t)\exp\left(-\frac{\|r_c - r_i\|^2}{2N_c^2(t)}\right) \qquad r_i \in \Re^2$$

$$h_{ci}(t) = \alpha(t), si \to c = i$$

Entrenamiento

- El entrenamiento de una red SOM se puede separar en distintas etapas:
  1. **Inicialización de parámetros y pesos.**
  2. **Presentación de un vector de entrada.**
  3. **Buscar neurona ganadora (best matching unit [bmu]).**
  4. **Actualización de pesos.**
  5. **Volver al paso 2 si no existen mas datos o se llego al número de iteraciones deseadas.**
- Existen 2 fases de entrenamiento en la red SOM.
  - La primera fase denominada **sintonización** la cual consiste en ordenar en forma global las neuronas en el mapa de datos, para una modelación correcta.
  - La fase de **afinamiento** la cual consiste en una sensibilización local de las unidades de salida ante las muestras.
- Ambas fases de entrenamiento se realizan de la misma manera, pero la inicialización de parámetros es distinta.

El Algoritmo

- Para toda las neuronas en la capa de salida: inicializar los pesos con valores aleatorios U(0,1)
- Si se trabaja con vectores normalizados, Normalizar vectores
- Elegir el modelo de aprendizaje (Ecuación diferencial)
- Elegir un modelo de vecino neuronal (función de feedback lateral).
- Elegir condición de parada.
- Construir a partir de la ED, la fórmula de adaptación de los pesos.
- Considerando tiempo discreto, repetir los pasos anteriores hasta que la condición de parada se cumpla:
  - Tomar la entrada x (t)
  - Para todas las neuronas j en la capa de salida, encontrar la ganadora.
  - Conociendo la ganadora, actualizar los pesos.

Dinámica de la Red

- **Función de aprendizaje de la red**:
  - El proceso de aprendizaje es no-supervisado.
  - El aprendizaje se desarrolla en tiempo discreto.
  - W = W (t)
  - En t = 0 los pesos son inicializados con valores aleatorios pequeños $W(0) = W_0$.
  - Los pesos se actualizan de la siguiente forma:
    - Para x(t) encontrar la neurona ganadora k.
    - Actualizar los pesos según modelo elegido:

$$\Delta W = W(t) - W(t-1) = (dW/dt)$$

Dinámica de la Red

- **Inicialización y condición de parada**:
  - Los pesos son inicializados con valores aleatorios pequeños.
  - La condición de parada del proceso de aprendizaje puede ser:
  - Elegir un número fijo de pasos.
  - El proceso de aprendizaje continúa hasta que la cantidad de ajuste: $\Delta w_{ji} = w_{ji}(t+1) - w_{ji}(t) \leq \varepsilon$

Parámetros

- Aunque no existe una ecuación para determinar el número de neuronas necesarias para modelar un problema, es obvio mencionar que no pueden haber más neuronas que datos.
- Luego de haber creado la malla es necesario inicializar el peso de cada neurona i ($\underline{W}_i = w_{1i}, w_{2i}, \ldots, w_{ni}$ con i = 1,2,...,k) con un valor aleatorio pequeño, además existen otros métodos de inicialización como fijar un valor predeterminado o comenzar con pesos nulos.
- Aunque no existe un número fijo de iteraciones, si se recomienda un número razonablemente grande de iteraciones por neurona.
- Otro parámetro a determinar es la tasa de aprendizaje a(t), la cual se mostrará mas adelante.

Actualización de pesos

- Una vez encontrada la neurona ganadora es necesario actualizar el peso de todas las neuronas, siendo la mas afectada, la neurona ganadora y sus vecinas próximas determinadas por la función de vecindad correspondiente:

$$\underline{w}_j(t+1) = \underline{w}_j(t) + \alpha(t)h\,(\underline{j}, \underline{w}_{bmu})[\underline{x}(t) - \underline{w}_j(t)] \quad j = 1 \ldots K$$

Tasa de aprendizaje

Función de vecindad

Tasa de Aprendizaje

- La tasa de aprendizaje α(t) es una función monótonamente decreciente con el tiempo que varia entre 0 y 1, que indica que tanto afecta la actualización del peso.
- En la fase de sintonización es recomendable comenzar con un valor alto y terminar cercano a 0.1, permitiendo con esto un ajuste global de las neuronas, mientras que en la fase de afinamiento los valores iniciales y finales varían entre 0.1 y 0.01.

$$\alpha(t) = \alpha_0 \left(\frac{\alpha_f}{\alpha_0} \right)^{t/t_\alpha}$$

$$\alpha(t) = \alpha_0 + (\alpha_f - \alpha_0)\, t / t_\alpha$$

$t_\alpha = número\ total\ de\ iteraciones$

$\alpha_0 = \alpha\ inicial\ (\leq 1)$

$\alpha_f = \alpha\ final\ (\cong 0.01)$

Función de Vecindad

- La vecindad en una red SOM se refiere al conjunto de neuronas vecinas que son afectadas por la neurona ganadora en la fase de entrenamiento.
- Existen distintos tipos de funciones de vecindad, como el sombrero mejicano la cual acerca a las neuronas cercanas y aleja a las restantes, pero existen otras que simplemente acercan a las neuronas cercanas sin influir en las lejanas, aunque no existe alguna función mejor que otra frecuentemente se usa la función (hiper)escalón simétrica.

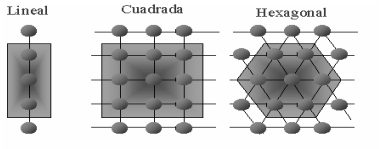
Función de Vecindad

- A medida que el número de iteraciones avanza en el entrenamiento la vecindad se hace más pequeña hasta afectar solamente a la neurona ganadora.
- En la fase de sintonización se usa una gran vecindad para poder ajustar en forma rápida las neuronas, mientras que en la fase de afinación la vecindad se fija en uno para ajustar solamente a las neurona ganadora.
- A continuación dos funciones de vecindad y una vista gráfica de cómo es su funcionamiento.

Función de Vecindad

Funciones de vecindad

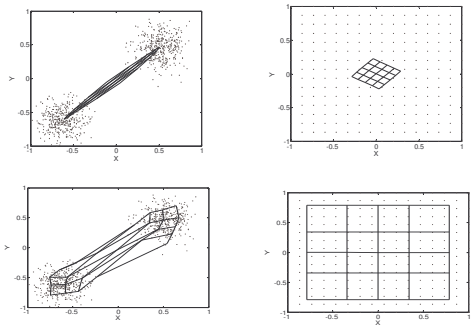
Escalón $h(\underline{j},\underline{w}_w)=\begin{cases}0, & si\ d(\underline{j},\underline{w}_{bmu})>R(t)\\ 1, & si\ d(\underline{j},\underline{w}_{bmu})\le R(t)\end{cases}$ $\quad R(t)=R_0+(R_f-R_0)t/t_r$



Gaussiana $h(\underline{j},\underline{w}_w)=\exp(-\|\underline{j}-\underline{w}_{bmu}\|^2/2\sigma(t)^2)$

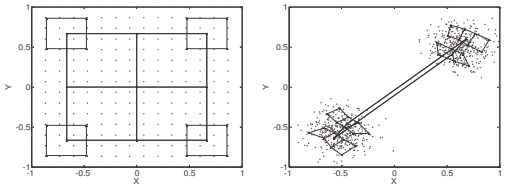
$\sigma(t)=$ *función monótonamente decreciente*

Self Organizing Maps (Ejemplos)



Problemas

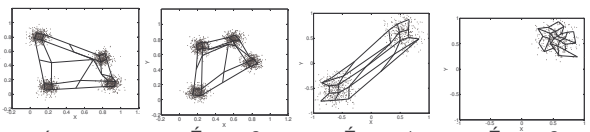
- Existen diversos problemas en las redes SOM que ya han sido solucionados con nuevos modelos, como GHSOM, el cual crece y jerarquiza los datos.



- Lamentablemente hay problemas que no han sido solucionados, encontrándose entre ellos, Interferencia Catastrófica y Robustez.

Interferencia Catastrófica

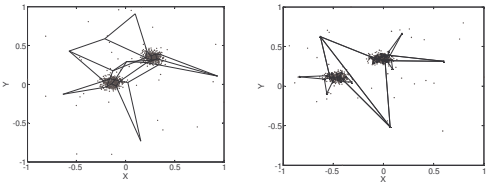
- El problema de Interferencia Catastrófica sucede cuando la información que fue recientemente aprendida a menudo elimina la que fue anteriormente aprendida. El problema principal consiste en diseñar un sistema que sea simultáneamente sensible a, pero no radicalmente destructivo, a la nueva entrada.



Época 1 Época 2 Época 1 Época 2

Robustez

- En la data real existen diversos puntos denominados outliers los cuales difieren del común de los datos. Las SOM no son sensibles a la presencia de este tipo de datos, no logrando realizar una buena generalización del conjunto de entrada, siendo otro problema importante que se debe abordar



Data con 5% de Outliers

Algunas aplicaciones de la SOM

- Data Mining en meteorología

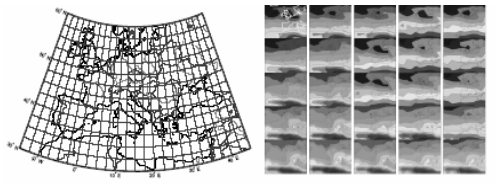


Fig. 1. (left) European region of ERA reanalysis 2.5° grid. (right) Surface temperature fields corresponding to 25 prototypes obtained with the SOM algorithm. For instance the upper-left prototype shows a isolated cold mass of air over Britain.

Consejos Prácticos

- Se recomiendan vecindades hexagonales
- Los datos se pueden emplear en forma cíclica, o bien en orden aleatorio (bootstrap learning)
- Escalamiento de los atributos de los datos de entrada, normalizados de acuerdo a sus varianzas

Medidas de desempeño

- Una medida empleada para la precisión, es el error de cuantificación medio EQM

$$EQM = \frac{1}{n}\sum_{i=1}^{n}(\|x_i - w_{bmu}\|^2)$$

- La preservación de la topología de los datos se puede evaluar

$$ETO = \frac{1}{n}\sum_{i=1}^{n}u(x_i)$$

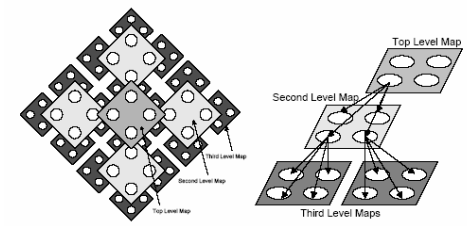
$$u(x_i) = 1 : d_i \in N(c)$$
$$u(x_i) = 0 : d_i \notin N(c)$$

Donde c es el BMU y d es el segundo lugar, para un input x

Limitaciones de la SOM

- Datos dentro de un mismo cluster pueden verse similares, pero puede haber una jerarquización de los datos
- Pueden existir clusters con una gran cantidad de datos, modelados por pocas neuronas.
- Las redes no son robustas en la presencia de outliers (se pueden usar métricas más robustas como por ejemplo la distancia de Mahalanobis)

GROWING HIERARCHICAL SELF-ORGANIZING MAP (GHSOM)



GROWING HIERARCHICAL SELF-ORGANIZING MAP (GHSOM)

- Este método es capaz de ajustar automáticamente su crecimiento y jerarquización en forma adaptativa a los datos.
- En el nivel 0 se forma una capa donde se encuentran todos los datos de input.
- Inicialmente en el nivel 1, se construye una SOM inicialmente de 2 por 2. Se aplica el algoritmo de SOM y esta red va creciendo iterativamente, de la siguiente manera:

GROWING HIERARCHICAL SELF-ORGANIZING MAP (GHSOM)

C_i = Subconjunto de vectores de entrada x, "mejor representados" por la unidad i

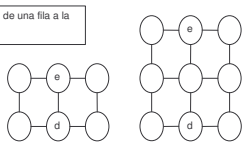
e=Unidad de error

$$e = \arg\max_i \left(\sum_{x_j \in C_i} \left\| m_i - x_j \right\| \right)$$

$$d = \arg\max_i \left(\left\| m_e - m_i \right\| \right)$$

$$m_i \in N_e$$

Inserción de una fila a la SOM



GROWING HIERARCHICAL SELF-ORGANIZING MAP (GHSOM)

Criterio de detención de crecimiento de un solo mapa SOM

MQE_m = Error de cuantización medio del **mapa** (solo el considerado actualmente)

qe_u = Error de cuantización de la neurona de la capa superior correspondiente.

Criterio de detención

$$MQE_m < \tau_1 \times qe_u$$

Donde $0 < \tau_1 < 1$ corresponde a un parámetro

GROWING HIERARCHICAL SELF-ORGANIZING MAP (GHSOM)

- Crecimiento Jerárquico

Sea m_0 la media de todos los datos de input
I = Conjunto de todos los datos de input

$$qe_0 = \sum_{x_i \in I} \left\| m_0 - x_i \right\|$$

Luego una neurona i no genera una SOM en el nivel inferior si:

$$qe_i < \tau_2 \times qe_0$$

GROWING HIERARCHICAL SELF-ORGANIZING MAP (GHSOM)

- De acuerdo a la elección de τ_1 se produce un trade-off entre los niveles de jerarquía y el tamaño de las SOM.
- La capacidad computacional requerida se reduce en la medida que se avanza en la jerarquía
- Cada SOM es independiente, y se entrena de acuerdo al algoritmo básico de SOM

Flexible Architecture of Self Organizing Maps

Rodrigo Salas, Héctor Allende,

Sebastián Moreno and Carolina Saavedra

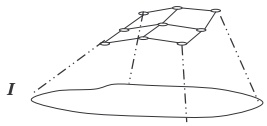
e-mail: rodrigo.salas@uv.cl

Topics

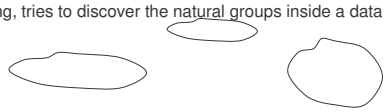
1. **Introduction**
2. **Changing Environments and Catastrophic Interference.**
3. **The Flexible Architecture of Self Organizing Maps**
4. **Evaluation of the adaptation Quality**
5. **Experimental Results**
6. **Concluding Remarks and Further Works**

1. Introduction

- **Self Organizing Maps (SOM)** are a very popular class of unsupervised neural networks that project high-dimensional data of the input space onto a neuron position in a low-dimensional output space grid.

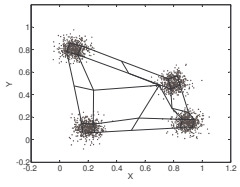


- A **cluster** is a collection of "similar" objects and they should be "dissimilar" to the objects belonging to other clusters. Unsupervised clustering, tries to discover the natural groups inside a data set.



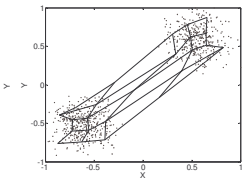
2. Changing Environments

- Under non-stationary environments the classification task changes during the operation of the model.
- Everything that exists changes with time. The changes could be minor fluctuations of the underlying probability distributions, steady trends, random or systematic, rapid substitution of one classification task with another and so on.



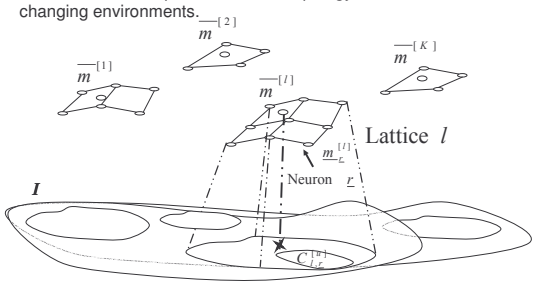
2. Catastrophic Interference

- Catastrophic Interference is a well known problem of Artificial Neural Networks (ANN) learning algorithms where the ANN forget useful knowledge while learning from new data.
- Furthermore the structure of most neural models must be chosen in advance.



3. The Flexible Architecture of Self Organizing Maps

- We present a hybrid model called Flexible Architecture of Self Organizing Maps (FASOM) that overcomes the Catastrophic Interference and preserves the topology of clustered data in changing environments.



3. The Flexible Architecture of Self Organizing Maps

- The FASOM is a hybrid model that adapts K receptive fields of dynamical self organizing maps and learn the topology of partitioned spaces.
- It has the capability of detecting novel data or clusters and creates new maps to learn this patterns avoiding that other receptive fields catastrophically forget.

- **First Part: Topological Learning Algorithm**
  - First Step: Clustering the data
  - Second Step: Topological Learning
  - Third Step: Growing the Maps Lattices
- **Second Part: Adapting to changing environments**
  - First step: Detection of novel data
  - Second Step: Creating Maps for the novel data
  - Third step: Integration of the maps
  - Fourth step: Learning the samples
  - Fifth step: Gradually forgetting the old data

3.1. First Part: Topological Learning Algorithm

- First Step: Clustering Data
  - We executed **K-means** algorithm, with a very low threshold in order to find a bigger number of clusters than they really are.
  - Then, the **Single Linkage** algorithm is executed to merge clusters that are closer and obtain the optimal number of clusters.
  - Results:
    - Number of clusters K
    - Centroids of clusters $\overline{m}^{[k]}$, where k = 1..K,

3.1. First Part: Topological Learning Algorithm

- Second Step: Topological Learning
  - Create a grid of size 2 x 2 in each cluster identified in the previous stage.
  - Search the best matching map (bmm)

Best matching units (bmu)

$$\eta = \arg\min_{k=1..K}\{\left\|\underline{x} - \underline{m}_{c_k}^{[k]}\right\|, \underline{m}_{c_k}^{[k]} \in \mathcal{M}_k\}$$

  - Update the weights of the map units

Neighborhood Function

$$\underline{m}_j^{[\eta]}(t+1) = \underline{m}_j^{[\eta]}(t) + \alpha(t) h_{c_\eta}^{[\eta]}(j,t)[\underline{x} - \underline{m}_j^{[\eta]}(t)] \qquad j = 1..M_\eta$$

Learning rate parameter

3.1. First Part: Topological Learning Algorithm

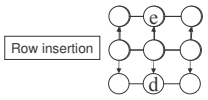
- **Third Step: Growing the Maps Lattices**
  - The unit with the highest quantization error, called error unit e, and its most dissimilar neighbor d are determined :

$$e = \arg\max_j \left\{ \sum_{\underline{x}_i \in C_j^{[u]}} \left\|\underline{x}_i - \underline{m}_j^{[u]}\right\| \right\}$$

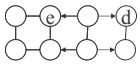
$$d = \arg\max_j \left(\left\|\underline{m}_e^{[u]} - \underline{m}_{l,j}^{[u]}\right\| \right), \quad \underline{m}_{l,j}^{[u]} \in \mathcal{N}_e$$

$\mathcal{N}_e$ is the set of neighboring units of the error unit *e*

  - A row or column of units is inserted between e and d.



Row insertion



Column insertion

  - After insertions, the map is trained again

3.2. Second Part: Adapting to changing environments

- First Step: Detection of novel data. $\chi_T^{[new]}$
  - Influence of the sample x to the model $FASOM_{T-1}$

$$\rho(\underline{x}, FASOM_{T-1}) = \left\|\underline{x} - \underline{m}_{c_\eta}^{[\eta]}(t)\right\|$$

- Second Step: Creating Maps for the novel data.
  - A new model $FASOM_T^{[new]}$ based on the samples extracted from the previous step is created
- Third step: Integration of the maps.
  - Both models are integrated in an unique updated version

$$FASOM_T = FASOM_{T-1} \bigcup FASOM_T^{[new]}$$

- Fourth step: Learning the samples.
  - The model FASOM learns the whole dataset.

3.2. Second Part: Adapting to changing environments

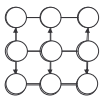
- Fifth step: Gradually forgetting the old data.
  - The maps gradually forget the clusters by shrinking their lattices towards their respective centroids and reducing the number of their prototypes.

$$\underline{m}_j^{[\kappa]}(T+1) = \underline{m}_j^{[\kappa]}(T) + \gamma[\underline{x} - \overline{m}^{[\kappa]}] \qquad j = 1..M_\kappa$$

Maps centroids

  - Maps contraction

Row elimination



$$\nu = \arg\min_{e=1..N_r-1; d=1..N_c-1} \left(\frac{1}{N_c}\sum_{j=1}^{N_c} \left\|\underline{m}_{(e,j)}^{[\kappa]} - \underline{m}_{(e+1,j)}^{[\kappa]}\right\|, \frac{1}{N_r}\sum_{i=1}^{N_r} \left\|\underline{m}_{(i,d)}^{[\kappa]} - \underline{m}_{(i,d+1)}^{[\kappa]}\right\| \right)$$

4. Evaluation of the adaptation Quality

To evaluate the clustering performance we compute the percentage of right classification given by

$$PC = \frac{1}{N}\sum_{\underline{x}_j,\, j=1\ldots N} u_{jk^*}$$

k^* = true class of $\underline{x}_j$

$$u_{jk} = \begin{cases} 1 & \text{if } x_j \in C_k \\ 0 & \text{otherwise} \end{cases}$$

To evaluate the quality of the partitions topological representation, we use the mean square quantization error

$$MSQE = \frac{1}{N}\sum_{M_k, k=1\ldots K} \sum_{\underline{m}_j^{[k]} \in M_k} \sum_{\underline{x}_i \in C_j^{[k]}} \left\|\underline{x}_i - m_j^{[k]}\right\|^2$$

References

[1] Kohonen, T., Hynninen, J., Kangas, J., Laaksonen, J. (1996). SOM_PAK: The self-organizing map program package. Report A31. Helsinki University of Technology, Laboratory of Computer and Information Science, Espoo, Finland. Also available in the Internet at the address http://www.cis.hut.fi/research/som_lvq_pak.shtml.

[2] Kohonen, T. (1995). *Self-Organizing Maps*. Series in Information Sciences, Vol. 30. Springer, Heidelberg. Second ed. 1997.