Universidad
Carlos III de Madrid

Simulador de mapas autoorganizados de Kohonen

Trabajo de Fin de Grado

Grado en Ingeniería Informática

24/09/2014

Autor: Juan Pedro García Ruiz
Tutor: José María Valls Ferrán

Título: Simulador de mapas autoorganizados de Kohonen

Autor: Juan Pedro García Ruiz 100283071

Tutor/Director: José María Valls Ferrán

Tribunal

Presidente: Ricardo Aler Mur

Secretario: Iván Vidal Fernández

Vocal: Ana María Iglesias Maqueda

Suplente: Agustín Orfila Díaz-Pabón

Realizado el acto de defensa y lectura del Trabajo de Fin de Grado el día 8 de octubre de 2014 en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de

Presidente

Secretario

Vocal

Resumen

En este proyecto, se ha desarrollado una aplicación que permite trabajar con los mapas autoorganizados de Kohonen, uno de los modelos de redes de neuronas más utilizados en aprendizaje no supervisado. Aunque el funcionamiento de este tipo de redes está perfectamente descrito mediante ecuaciones matemáticas, no siempre es fácil entenderlo desde un punto de vista intuitivo.

Una importante motivación para la elaboración de este proyecto era la de proporcionar información visual que permitiese entender intuitivamente el funcionamiento de este tipo de redes. La forma de hacerlo es mostrar gráficamente cómo se realiza el entrenamiento en situaciones muy sencillas: cuando los datos tienen solamente dos dimensiones. Esta es una forma efectiva de ayudar a comprender el comportamiento de estas redes y cómo influyen en el mismo los diferentes parámetros. Además, el usuario puede crear el conjunto de datos de entrenamiento fácilmente con clics de ratón, regular la velocidad a la que transcurre el entrenamiento y pausarlo si lo desea, configurar los parámetros, etc.

Sin embargo, aunque didácticamente es algo muy útil, el entrenamiento con datos bidimensionales no sirve para resolver problemas reales, ya que en éstos los datos tienen más dimensiones. La aplicación desarrollada también es capaz de trabajar con más dimensiones, aunque sin proporcionar información visual sobre los datos. De esta forma, también es apta para resolver problemas reales e, incluso, realizar aprendizaje supervisado. Para esto último, la aplicación cuenta con la capacidad de realizar una calibración sobre el mapa una vez finalizado el entrenamiento. Esto consiste en tener en cuenta las clases de los datos para saber cómo de bien los clasifica el mapa, entrenado de forma no supervisada.

En todo tipo de casos, el usuario podrá ver en una gráfica la evolución del error cometido por la red durante el entrenamiento, una vez que el mismo ha finalizado.

Además, la aplicación puede utilizarse en diferentes sistemas operativos, siempre y cuando tengan instalada la máquina virtual de Java o JVM (Java Virtual Machine).

Abstract

In this project, an application has been developed to work with Kohonen Self-Organizing Maps (SOM), one of the most commonly used neural network models for unsupervised machine learning. Although this kind of neural network model is defined perfectly by mathematical equations, it is not always easy to understand from an intuitive point of view.

An important motivation for the development of this project was to provide visual information to understand intuitively how this kind of network works. The way to do it is to show graphically how training is done in very simple situations: when input patterns are two-dimensional. This is an effective way to help understand the behavior of these networks and how the parameters influence. In addition, the user can create a training dataset easily with mouse clicks, adjust the training speed and pause it if he wants, configure parameters, etc.

However, although didactically is very useful, training with two-dimensional data does not solve real problems because in those cases patterns are in a space with more dimensions. The developed application is also able to work with more dimensions, even though without providing visual information about data. Thus, it is also possible to solve real problems and even perform supervised learning. For the latter, the application has the ability to perform a calibration on the map after the training ended. Calibration is to take into account the classes of the patterns to check if map is classifying rightly. But map training is always unsupervised.

In any case, the user can see a graph with the evolution of the error made by the network during training, once it has finished.

In addition, the application can be used in many operative systems, if they have the JVM (Java Virtual Machine) installed.

Índice general

Índice de ilustraciones

Índice de tablas

1. Introducción

1.1. Motivación

Las redes de neuronas son modelos matemáticos que se construyen a partir de ejemplos y permiten resolver gran cantidad de problemas reales. En las enseñanzas de Ingeniería Informática existen materias como Inteligencia Artificial y Redes de Neuronas Artificiales donde los alumnos deben comprender con detalle el funcionamiento de estos modelos de Redes de Neuronas. Aunque están perfectamente descritos por medio de ecuaciones matemáticas, también requieren una comprensión más intuitiva. En general, los profesores se encuentran con la dificultad de transmitir al alumno una visión intuitiva de estos modelos, es decir, que los alumnos imaginen y visualicen mentalmente qué ocurre cuando una red de neuronas se está entrenando.

Es importante entender el significado de las ecuaciones matemáticas que describen los modelos, pero también es fundamental comprender intuitivamente cómo funcionan estos modelos y esto se facilita enormemente si se puede **visualizar** lo que ocurre durante el proceso de construcción de los modelos, es decir durante el entrenamiento de estas redes de neuronas. Por supuesto, esta visualización sólo se podrá hacer en ciertas situaciones simplificadas, donde los datos se puedan representar en dos dimensiones y no sean excesivamente numerosos, pero permite al alumno entender intuitivamente su fundamento y no representará ninguna dificultad "imaginar" el mismo modelo extendido a espacios de datos con más dimensiones.

En conversaciones con profesores de esta materia, me han transmitido la necesidad de una aplicación sencilla e intuitiva que permita visualizar diferentes partes del aprendizaje de redes de neuronas, y particularmente con un tipo especial y muy importante de red, los mapas autoorganizados de Kohonen, muy utilizados en problemas donde se requiere un aprendizaje no supervisado. Yo mismo he llegado a pensar que las ecuaciones matemáticas resultaban muy complicadas, por no saber qué significaba cada uno de sus componentes ni por qué se relacionaban como lo hacían, pero una vez entendida la idea intuitiva y cómo se comporta el mapa, es más sencillo entender y aplicar las ecuaciones. Además, cuando se aplican de forma errónea, a veces es posible identificarlo en base a los resultados y a la idea intuitiva sobre cómo se entrenan.

No es fácil encontrar software que permitan visualizar este proceso, porque existen algunos *applets* que permiten cierta visualización pero son muy básicos y no tienen en cuenta algunos de los parámetros más relevantes.

Esta es la motivación fundamental para desarrollar este trabajo. Además, también debe servir para trabajar con datos reales de más dimensiones, aunque no se puedan visualizar.

1.2. Objetivos

1.2.1. Objetivos generales

A continuación, se mencionan algunos objetivos generales de la aplicación, relacionados con las ideas que motivaron la misma:

- La aplicación que se implemente debe poder servir como herramienta didáctica para visualizar lo que ocurre cuando se entrenan los mapas autoorganizados de Kohonen.
- La aplicación también servirá para ser utilizada con datos reales con cualquier número de dimensiones, aunque no se puedan visualizar.

1.2.2. Objetivos parciales

A continuación, se exponen algunos objetivos parciales que deberá cumplir la aplicación, relacionados con su implementación:

- La aplicación debe tener un uso sencillo e intuitivo. El usuario podrá situar los datos en el plano de forma sencilla con el ratón.
- La aplicación debe ser robusta: en caso de error debe informar al usuario de ello sin dejar de funcionar.
- Aunque el algoritmo de entrenamiento es no supervisado, a veces los datos reales están etiquetados. En este caso puede aprovecharse esta situación para clasificar nuevos datos. Esta característica se contemplará en esta aplicación.
- La aplicación será válida para diferentes sistemas operativos (Windows y Linux).
- Una vez construido el modelo utilizando los datos de entrenamiento, podrá validarse usando datos de test.
- Se mostrará la evolución de alguna medida de error a lo largo del entrenamiento.

1.3. Estructura de la memoria

En este apartado, se describe brevemente el contenido de cada uno de los apartados posteriores de esta memoria.

En el apartado “2. Planteamiento del problema” se comenzará realizando un análisis del estado del arte, dando una descripción detalla sobre qué son y cómo funcionan los mapas de Kohonen. A continuación, se hablará sobre algunos softwares disponibles para utilizar mapas de Kohonen y sobre Java, el lenguaje de programación elegido para desarrollar la aplicación. Después, se expondrán formalmente los requisitos que deberá tener la aplicación. Por último, se explicará cuál es el marco regulador que afecta al desarrollo del proyecto.

En el apartado “3. Diseño de la solución técnica”, se detallan diferentes aspectos técnicos relacionados con la implementación de la aplicación y se adjunta un diagrama de clases con una descripción del mismo.

En el apartado “4. Resultados y evaluación” se habla sobre los principales intereses que puede tener un usuario, las pruebas realizadas y sus resultados, así como su influencia en el resultado de la aplicación en general. También se muestran ejemplos de uso de la aplicación.

En el apartado “5. Planificación y presupuesto del trabajo” se detalla una planificación temporal sobre el proyecto, dividida en diferentes fases. Después, se especifica cuál será el presupuesto necesario para el desarrollo de la aplicación, teniendo en cuenta los diferentes costes que ocasiona y la planificación.

En el apartado “6. Conclusiones” se hará una valoración del desarrollo de todo el proyecto. Se tendrá en cuenta cuáles han sido los resultados del mismo en relación a lo que se pretendía en un principio. También se propondrán mejoras que podrían añadirse en un futuro.

En el apartado “7. Anexos” se incluirán más detalles del diagrama de clases, así como un manual de usuario de la aplicación. También se incluye la parte en inglés.

En el apartado “8. Referencias”, simplemente se encuentra el conjunto numerado de referencias usadas durante el desarrollo del proyecto.

2. Planteamiento del problema

2.1. Análisis del estado del arte

2.1.1. Descripción de los mapas de Kohonen

Un mapa de Kohonen o mapa autoorganizado (SOM –Self-Organizing Map–) es un tipo de red de neuronas artificiales que realiza aprendizaje no supervisado. Debe su nombre a su creador, Teuvo Kohonen (nacido el 11 de julio de 1934), Profesor Emérito de la Academia de Finlandia. Ha publicado más de 300 artículos de investigación sobre redes de neuronas y reconocimiento de patrones y 5 libros. [4]

Los mapas de Kohonen pueden usarse en la práctica para abordar diferentes problemas, entre otros:

- agrupamiento o *clustering* [2]
- minería de datos (*data mining)*
- análisis, diagnóstico, monitorización y control de procesos
- aplicaciones biomédicas, incluyendo métodos de diagnóstico y análisis de datos en bioinformática
- análisis de datos en comercio, industria, macroeconomía y finanzas. [3]

Este tipo de red de neuronas artificiales tiene dos capas: la capa de entrada (o F1) y la capa de competición (o F2).

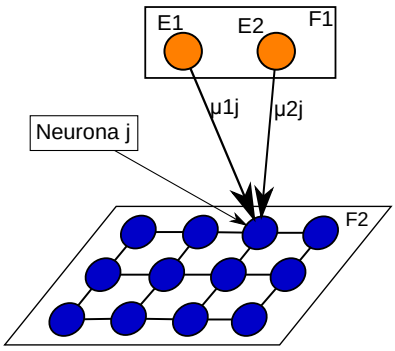


Ilustración 1: red de neuronas con un mapa de 3×4 con topología rectangular y datos de dos dimensiones (dos atributos)

La capa de entrada tiene tantas neuronas como atributos tienen los datos de entrada. Cada neurona está conectada con todas las neuronas de la capa de competición a través de unos pesos (existe un peso entre cada neurona de la capa de entrada y cada neurona de la capa de competición). Cada dato o patrón de entrada se transmite desde la capa de entrada a la de competición, donde se calcula la activación de cada neurona para ese dato. Los datos de entrada deben ser vectores numéricos, es decir, cada componente del vector (atributo) debe ser un número real.

La capa de competición suele tener una estructura bidimensional. La cantidad de neuronas que tiene es un parámetro que hay que especificar en la creación de la red, concretando cuántas filas y cuántas columnas tiene. A cada neurona le llega la señal de la capa de entrada y se calcula su activación. La que tiene la activación más alta en toda

la capa es denominada "ganadora" y es la neurona cuyos pesos son actualizados, y también los de sus vecinas, siguiendo una ley de aprendizaje.

La función de vecindario es una función que dados una neurona, una función de distancia y un límite máximo de vecindario, indica cuáles son las neuronas vecinas de la dada y a qué distancia están. Generalmente se usa como función de distancia la distancia euclídea. El concepto de vecindario es muy importante en los mapas de Kohonen, ya que el hecho de que unas neuronas sean vecinas de otras y que cuando una neurona cambia sus vecinas también lo hagan es lo que hace que este tipo de red tenga propiedades tan interesantes. El límite de vecindario es dinámico: va decreciendo a medida que avanza el entrenamiento (puede decrecer de diferentes formas: lineal, exponencial…).

La disposición de las neuronas en el mapa es algo que influye en el vecindario, siendo las topologías más habituales la rectangular y la hexagonal:

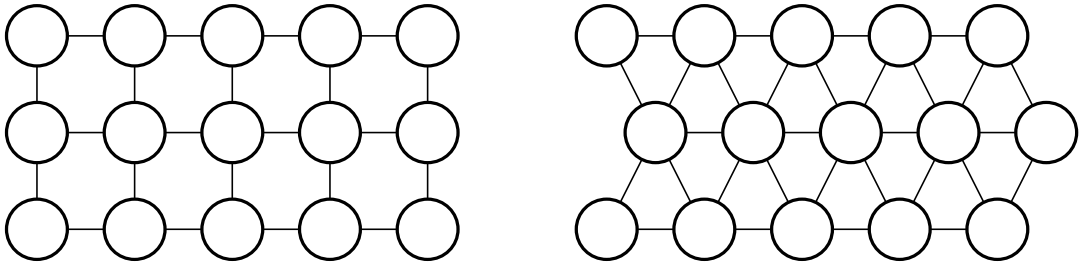


Ilustración 2: dos mapas de 3×5 con diferentes topologías: a la izquierda con topología rectangular y a la derecha hexagonal

Existe otro parámetro de la red denominado tasa o razón de aprendizaje. Se trata de un número real en el rango [0,1] que indica cuánto varían los pesos de las neuronas cuando deben ser actualizados (por haber sido ganadoras o vecinas de ganadoras). Al igual que el límite de vecindario, la tasa de aprendizaje va decreciendo a lo largo del entrenamiento. Que ambos parámetros vayan decreciendo implica que, si no comienzan con valores excesivamente bajos, el mapa sufra cambios bruscos durante los primeros ciclos pasando por variaciones más moderadas a medida que el entrenamiento va avanzando terminando con ligeras variaciones cuando el entrenamiento está llegando a su fin.

La ley de aprendizaje que se aplica sobre las neuronas es la siguiente:

$$\Delta\mu_{ij} = \begin{cases} \dfrac{\alpha(t)}{\sigma(c_k, c_j)}\left(e_i(t) - \mu_{ij}(t)\right), \text{si } c_k \text{ es la ganadora y } \sigma(c_k, c_j) \leq \theta(t) \\ 0, \text{en caso contrario} \end{cases}$$

donde:

- μ_{ij} es el peso i de la neurona j. Las neuronas tienen tantos pesos como atributos tienen los datos de entrada.
- $\alpha \in [0,1]$ es la tasa o razón de aprendizaje.
- $\sigma(c_k, c_j)$ es el grado de vecindad entre las neuronas i y j. Se define como: $\sigma(c_k, c_j) = 1 + d(c_k, c_j)$ siendo $d(c_i, c_j)$ la distancia entre las neuronas i y j según la función de distancia elegida. En el caso de la euclídea:

$d(c_k, c_j) = \sqrt{\sum_{i=1}^{n}(\mu_{ik} - \mu_{ij})^2}$, siendo n la cantidad de atributos de los datos de entrada.

- e_i es el atributo i del patrón de entrada e.
- $\theta \in \mathbb{R}^+ \geq 1$ es el límite de vecindario.

El entrenamiento de un mapa de Kohonen podría sintetizarse en los siguientes pasos:

1. Inicializar los pesos con valores aleatorios pequeños.
2. Presentar un patrón de entrada.
3. Calcular la distancia del patrón presentado a cada neurona de la red.
4. Seleccionar la neurona ganadora, la más cercana al patrón presentado.
5. Obtener las vecinas de la ganadora y sus distancias a ella.
6. Aplicar la ley de aprendizaje.
7. Volver al paso 2, hasta haber presentado el conjunto completo de patrones de entrada.
8. Actualizar la tasa de aprendizaje (α) y el límite de vecindario (θ).
9. Si se han hecho menos ciclos de los indicados, volver al paso 2. En caso contrario, fin.

Dado que el tipo de aprendizaje que realizan estas redes es no supervisado, no se puede hablar del error cometido de la misma manera que se hace en el aprendizaje supervisado, en el que se cuenta con una salida deseada. A pesar de ello, es posible medir cómo se está realizando el ajuste de las neuronas a los datos, ya que el algoritmo minimiza la distancia entre cada neurona y el conjunto de datos al que representa. Sabiendo esto, puede comprobarse cuál es la distancia entre cada patrón y la neurona que lo representa y tener en cuenta todas esas distancias, ya sea sumándolas todas, haciendo una media por cada neurona, etc.

Aunque, como ya se ha comentado anteriormente, este tipo de red de neuronas realiza aprendizaje no supervisado, es posible utilizarlas para realizar aprendizaje supervisado. Para ello, se debe calibrar el mapa una vez finalizado el entrenamiento.

La calibración consiste en asignar una clase a cada neurona. Para saber qué clase asignar a una neurona, se comprueba qué datos de todo el conjunto de entrenamiento tienen a dicha neurona por prototipo que los representa y qué clases tienen. La clase que más veces se repita en dicho conjunto de datos será la que se le asigne a la neurona.

Es muy importante aclarar que el método de aprendizaje utilizado durante el proceso de entrenamiento es **no** supervisado y que la calibración del mapa se realiza una vez finalizado el mismo, sin haberle afectado en ningún momento de su transcurso. Tras realizar este etiquetado, no todas las neuronas tienen por qué tener una clase asignada. Es posible que haya neuronas "sueltas", sin patrones a los que representen y, por tanto, no será posible asignarlas una clase. Aunque puede ocurrir que haya neuronas que no representan a ningún patrón, cada patrón es representado por exactamente una neurona.

Si tras realizar la calibración de un mapa se clasificara el conjunto de entrenamiento, cada patrón se clasificaría por la neurona más cercana y todos quedarían clasificados. Sin embargo, si se clasificara un conjunto de test, éste podría contener patrones cuya neurona más cercana fuera una no etiquetada en el proceso de calibración y, por tanto,

quedaría sin clasificar. Todos los patrones con los que se diera esta situación serían contabilizados como fallos, ya que no se les asigna clase pero realmente pertenecen a una.

Los pasos que podrían darse para realizar aprendizaje supervisado son los siguientes:

1. Configurar los parámetros del mapa y entrenarlo (este paso no tiene en cuenta que los datos están etiquetados).
2. Calibrar el mapa utilizando las clases de los datos.
3. Utilizando la calibración hecha en el paso 2, comprobar la tasa de aciertos de entrenamiento y de test.
4. Volver al paso 1 si se quieren probar distintos mapas.

[1][2]

2.1.2. Software para usar mapas de Kohonen

2.1.2.1. Simuladores para resolver problemas reales

En cuanto a software para utilizar este tipo de red de neuronas, cabe destacar "SOM_PAK", desarrollado por el Centro de Investigación de Redes de Neuronas Artificiales del Laboratorio de Computación y Ciencias de la Información de la Universidad de Tecnología de Helsinki.

Se trata de un software de dominio público [6] que, en lugar de presentarse como una sola aplicación, consta de varios programas que hay que usar a través de la línea de comandos y que en conjunto ofrecen toda la funcionalidad. Estos programas son, entre otros:

- randinit: sirve para inicializar aleatoriamente los vectores de pesos. Es necesario especificarle las dimensiones del mapa, un archivo de entrada, un archivo de salida, el tipo de vecindario y la topología del mapa.
- vsom: se encarga de entrenar el mapa. Es necesario especificarle la cantidad de ciclos, los valores iniciales de la tasa de aprendizaje y del límite de vecindario, así como un archivo de entrada, que es el de salida de randinit y otro de salida.
- vcal: calibra el mapa en función de las clases especificadas para cada patrón de entrada.
- qerror: cuantifica el error cometido por la red.
- sammon: genera una imagen (la proyección de Sammon) con la finalidad de poder ofrecer cierta información sobre los datos desde un punto de vista visual.

A continuación, se expone un ejemplo de uso sencillo, incluyendo los pasos básicos. Se debe tener en cuenta que el programa ofrece más opciones y para un uso completo debe consultarse la documentación del mismo.

El primer paso es inicializar el mapa:

```
> randinit -din train.dat -cout map.cod -xdim 12 -ydim 8
  -topol hexa -neigh bubble -rand 123
```

Este comando inicializa el mapa para entrenar con los datos del archivo "train.dat" y genera como salida el archivo "map.cod". Crea un mapa de 8 filas y 12 columnas ("-xdim" indica la dimensión en el eje X y "-ydim" en el eje Y). El mapa creado tiene

topología hexagonal, tipo de vecindario "bubble" y sus pesos se inicializan aleatoriamente usando como semilla el número 123.

El siguiente paso es el entrenamiento:

```
> vsom -din train.dat -cin map.cod -cout map.cod -rlen 1000
  -alpha 0.05 -radius 10
```

Se entrena el mapa con los mapas del archivo "train.dat", usando el mapa inicializado anteriormente "map.cod" y se guarda en "map.cod". El entrenamiento tiene 1000 ciclos o iteraciones y comienza con una razón de aprendizaje de 0.05 y una distancia de límite de vecindario inicial de 10.

Una vez realizado el entrenamiento, se puede cuantificar el error cometido con el conjunto de datos y el mapa entrenado:

```
> qerror -din train.dat -cin map.cod
```

Después, es posible calibrar el mapa si el conjunto de datos está etiquetado:

```
> vcal -din train.dat -cin map.cod -cout map.cod
```
[5]

2.1.2.2. Aplicaciones que ilustran el funcionamiento del SOM

Otras aplicaciones que utilizan mapas de Kohonen y que ofrecen visualización gráfica son algunos *applets*, bastante limitados en las opciones que ofrecen. Cabe destacar uno realizado por Maximilian Bügler, investigador en la Universidad Técnica de Munich. A continuación se muestra una imagen del *applet* en funcionamiento:

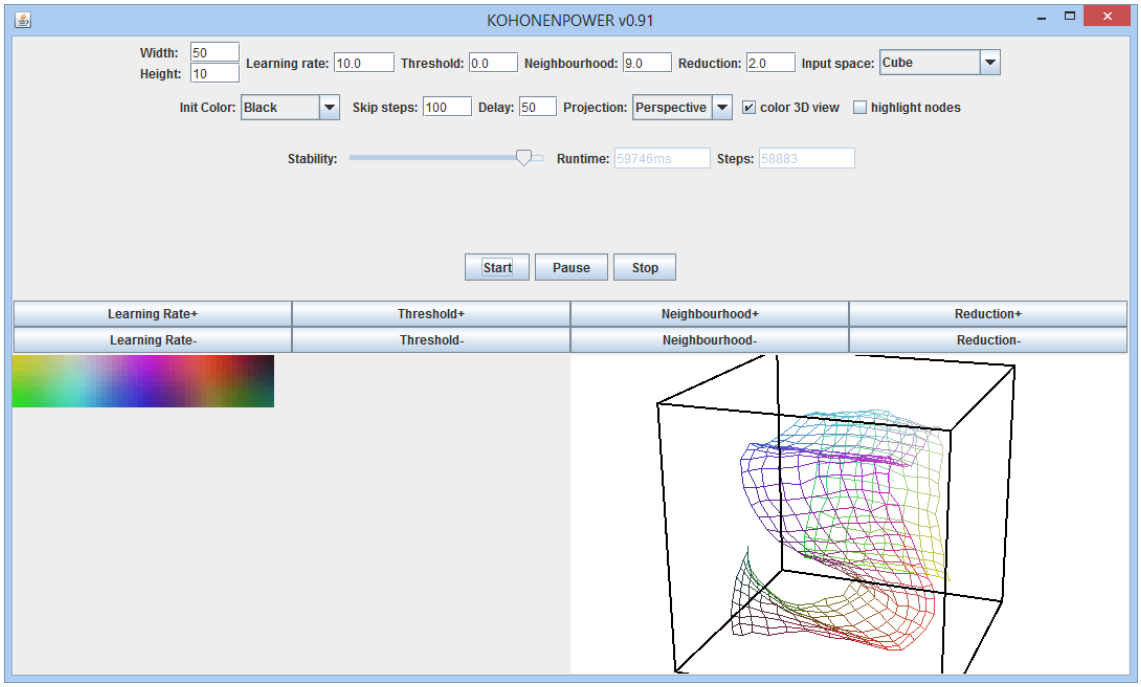


Ilustración 3: applet de KOHONENPOWER v0.91

La parte superior permite configurar los parámetros de la red. En la parte inferior derecha, se ve un cubo con una especie de malla de colores dentro. El espacio de entrada forma un cubo que se encontraría en el interior del cubo que se ve y la malla es el mapa, que tiende a ajustarse a ese cubo interior. Durante la ejecución, se va viendo cómo cambia a medida que avanza el entrenamiento. Puede verse como, por acción del vecindario, neuronas próximas entre sí en el mapa están próximas en el espacio de entrada.

Además de configurar los parámetros usuales de un mapa de Kohonen, el applet permitir decidir la forma geométrica del espacio de entrada, entre un conjunto de ellas donde se incluyen, entre otras, el cubo, la esfera y el plano.

2.1.3. Java

Java es un lenguaje de programación creado en 1991 por un pequeño grupo de ingenieros de la compañía Sun llamados el "Green Team", liderados por James Gosling, con la finalidad de actuar de nexo entre dispositivos digitales y ordenadores. [9]

Desde entonces, Java ha evolucionado mucho, al igual que lo ha hecho la tecnología, pero sin perder de vista el objetivo con el que fue creado. Tanto es así, que afirma que el lenguaje cumple con la frase "*write once, run anywhere"*. Quiere decir que con Java se escribe el código para una aplicación una vez y se puede ejecutar en cualquier dispositivo independientemente de su arquitectura y sistema operativo, mientras tenga instalada la Máquina Virtual de Java (JVM –Java Virtual Machine–), siendo de alguna manera una unión entre distintos dispositivos. [10]

Esta portabilidad y el gran desarrollo a lo largo del tiempo han hecho que Java se haya ido adaptando con éxito a la época actual, habiéndose convertido en un lenguaje de programación ampliamente utilizado:

- El 97% de los escritorios empresariales ejecutan Java
- El 89% de los escritorios (o computadoras) en Estados Unidos ejecutan Java
- 9 millones de desarrolladores de Java en todo el mundo
- 3 mil millones de teléfonos móviles ejecutan Java
- El 100% de los reproductores de Blu-ray incluyen Java
- 125 millones de dispositivos de televisión ejecutan Java [8]

No obstante, el uso de Java también tiene desventajas respecto a otros lenguajes de programación:

- Archivos binarios de otros lenguajes son más densos que los *byte-codes* que genera Java, lo que hace que descargar algo de Java a través de una red sea más lento.
- Java es menos flexible que otros lenguajes a optimizaciones de bajo nivel, cuya realización depende de la arquitectura del sistema concreto sobre el que se ejecutará la aplicación.
- El código optimizado compilado en C es considerablemente más rápido que los *byte-codes* generados con Java. [7]

2.1.4. Conclusión

A la hora de seleccionar la tecnología a utilizar, hay que tener en cuenta que en este trabajo se engloban los dos casos expuestos anteriormente: que el programa pueda usarse para resolver problemas reales y que sirva para visualizar el entrenamiento en casos sencillos, de dos dimensiones.

Se ha decidido usar Java por tener la ventaja de que la aplicación será válida para diferentes sistemas operativos sin necesidad de adaptaciones en el código para ello. Puesto que el objetivo principal es el de crear una herramienta didáctica, es conveniente que pueda ser usada por cualquier alumno, que probablemente preferirá usarla junto con material que tiene en su ordenador (apuntes, prácticas, consultas web, otras aplicaciones…) sin preocuparse de si será compatible con el sistema operativo sobre el que esté ya trabajando. Además, el conocimiento y experiencia adquiridos sobre Java es mayor que sobre otros lenguajes, lo que puede suponer menos contratiempos y retrasos.

En contraposición, el rendimiento en los entrenamientos de las redes será menor que el que podría haberse conseguido usando otros lenguajes, como C. Esta diferencia será más notable con entrenamientos que usen ficheros como entrada, especialmente cuanto mayor sea la cantidad de datos que contienen y mayor sea la cantidad de dimensiones.

No obstante, la parte que proporciona la visualización gráfica del entrenamiento es la que realmente cumple con el objetivo de enseñar cómo es el entrenamiento de las redes neuronales autoorganizadas. En este caso, los datos sólo tienen dos dimensiones y no es necesario que el entrenamiento se haga a gran velocidad, ya que será visto por una persona.

2.2. Requisitos

En este apartado, se especifica de manera formal cuáles deben ser las funcionalidades de la aplicación a desarrollar, por medio de requisitos.

Cada requisito se define en una tabla con los siguientes campos:

- Identificador (ID). Cadena de caracteres que identifica al requisito de forma unívoca. Su estructura es R<tipo>-<número>, donde:
  - Tipo. Es el tipo del requisito. Pueden ser:
    - Funcional (FU). Indica qué debe hacer la aplicación.
    - No funcional (NF). Indica cómo debe hacer algo la aplicación.
    - Negativo (NE). Indica qué no se puede hacer bajo ciertas condiciones.
  - Número. Conjunto de cifras para numerar los requisitos que comienzan en 001.
- Necesidad. Sirve para indicar cómo de importante es el requisito para garantizar la calidad de la aplicación. Puede ser:
  - Esencial
  - Deseable
  - Opcional
- Prioridad. Útil para establecer un orden de implementación de los requisitos. Puede tomar tres valores:
  - Alta

    - Media
    - Baja
- Fuente. Indica quién es la persona que dio origen al requisito, si el tutor o el alumno.
- Verificabilidad. Señala la facilidad con que es posible comprobar que la aplicación cumple el requisito. Puede ser:
    - Alta
    - Media
    - Baja
- Descripción. Se trata de una explicación breve, en la medida de lo posible, del requisito.

2.2.1. Requisitos funcionales

2.2.1.1. Relacionados con los posibles métodos de entrada

ID	RFU-001		
Necesidad	☒ Esencial	☐ Deseable	☐ Opcional
Prioridad	☒ Alta	☐ Media	☐ Baja
Fuente	☒ Tutor	☐ Alumno	
Verificabilidad	☒ Alta	☐ Media	☐ Baja
Descripción	La aplicación podrá recibir como entrada para entrenamiento un fichero de entrenamiento seleccionado por el usuario.		

Tabla 1: requisito funcional 1

ID	RFU-002		
Necesidad	☒ Esencial	☐ Deseable	☐ Opcional
Prioridad	☒ Alta	☐ Media	☐ Baja
Fuente	☒ Tutor	☐ Alumno	
Verificabilidad	☒ Alta	☐ Media	☐ Baja
Descripción	La aplicación podrá recibir como entrada para entrenamiento un conjunto de datos generado por el usuario en una gráfica.		

Tabla 2: requisito funcional 2

ID	RFU-003		
Necesidad	☒ Esencial	☐ Deseable	☐ Opcional
Prioridad	☒ Alta	☐ Media	☐ Baja
Fuente	☒ Tutor	☐ Alumno	
Verificabilidad	☒ Alta	☐ Media	☐ Baja
Descripción	La aplicación podrá recibir como entrada un fichero de test para evaluar su contenido tras el entrenamiento, seleccionado por el usuario.		

Tabla 3: requisito funcional 3

ID	RFU-004		
Necesidad	☒ Esencial	☐ Deseable	☐ Opcional
Prioridad	☒ Alta	☐ Media	☐ Baja
Fuente	☒ Tutor	☐ Alumno	
Verificabilidad	☒ Alta	☐ Media	☐ Baja
Descripción	La aplicación podrá recibir como entrada para entrenamiento un conjunto de datos generado por el usuario en una gráfica.		

Tabla 4: requisito funcional 4

2.2.1.2. Relacionados con el método de entrada gráfica

ID	RFU-005		
Necesidad	☒ Esencial	☐ Deseable	☐ Opcional
Prioridad	☒ Alta	☐ Media	☐ Baja
Fuente	☒ Tutor	☐ Alumno	
Verificabilidad	☒ Alta	☐ Media	☐ Baja
Descripción	Con el método de entrada gráfica, el usuario generará los datos de entrenamiento mediante clics con el botón izquierdo del ratón, siempre y cuando la red no esté entrenándose.		

Tabla 5: requisito funcional 5

ID	RFU-006		
Necesidad	☒ Esencial	☐ Deseable	☐ Opcional
Prioridad	☒ Alta	☐ Media	☐ Baja
Fuente	☒ Tutor	☐ Alumno	
Verificabilidad	☒ Alta	☐ Media	☐ Baja
Descripción	Con el método de entrada gráfica, el usuario podrá inicializar neuronas directamente sobre la gráfica		

Tabla 6: requisito funcional 6

ID	RFU-007		
Necesidad	☐ Esencial	☒ Deseable	☐ Opcional
Prioridad	☐ Alta	☒ Media	☐ Baja
Fuente	☒ Tutor	☐ Alumno	
Verificabilidad	☒ Alta	☐ Media	☐ Baja
Descripción	Con el método de entrada gráfica, el usuario podrá cambiar la posición de neuronas inicializadas sobre la gráfica.		

Tabla 7: requisito funcional 7

ID	RFU-008		
Necesidad	☐ Esencial	☒ Deseable	☐ Opcional
Prioridad	☐ Alta	☒ Media	☐ Baja
Fuente	☒ Tutor	☐ Alumno	
Verificabilidad	☒ Alta	☐ Media	☐ Baja
Descripción	Con el método de entrada gráfica, se mostrarán las coordenadas de cada neurona en el mapa, junto a la neurona.		

Tabla 8: requisito funcional 8

ID	RFU-009		
Necesidad	☐ Esencial	☒ Deseable	☐ Opcional
Prioridad	☐ Alta	☒ Media	☐ Baja
Fuente	☒ Tutor	☐ Alumno	
Verificabilidad	☒ Alta	☐ Media	☐ Baja
Descripción	Con el método de entrada gráfica, el usuario podrá elegir si se muestran o no las coordenadas de cada neurona.		

Tabla 9: requisito funcional 9

ID	RFU-010		
Necesidad	☒ Esencial	☐ Deseable	☐ Opcional
Prioridad	☒ Alta	☐ Media	☐ Baja
Fuente	☒ Tutor	☐ Alumno	
Verificabilidad	☒ Alta	☐ Media	☐ Baja
Descripción	Con el método de entrada gráfica, durante el entrenamiento, la gráfica se actualizará al menos una vez cada ciclo.		

Tabla 10: requisito funcional 10

ID	RFU-011		
Necesidad	☒ Esencial	☐ Deseable	☐ Opcional
Prioridad	☒ Alta	☐ Media	☐ Baja
Fuente	☒ Tutor	☐ Alumno	
Verificabilidad	☒ Alta	☐ Media	☐ Baja
Descripción	Con el método de entrada gráfica, durante el entrenamiento, el usuario podrá ver la siguiente información: • Ciclo actual • Tasa de aprendizaje del ciclo actual • Límite de vecindario del ciclo actual • Sumatorio de distancias euclídeas entre cada dato y la neurona en que se clasifica.		

Tabla 11: requisito funcional 11

ID	RFU-012		
Necesidad	☒ Esencial	☐ Deseable	☐ Opcional
Prioridad	☒ Alta	☐ Media	☐ Baja
Fuente	☒ Tutor	☐ Alumno	
Verificabilidad	☒ Alta	☐ Media	☐ Baja
Descripción	Con el método de entrada gráfica, el usuario podrá borrar el conjunto entero de datos, mientras la red no esté siendo entrenada.		

Tabla 12: requisito funcional 12

ID	RFU-013		
Necesidad	☒ Esencial	☐ Deseable	☐ Opcional
Prioridad	☒ Alta	☐ Media	☐ Baja
Fuente	☒ Tutor	☐ Alumno	
Verificabilidad	☒ Alta	☐ Media	☐ Baja
Descripción	Con el método de entrada gráfica, el usuario podrá borrar la red de neuronas, mientras no esté siendo entrenada.		

Tabla 13: requisito funcional 13

ID	RFU-014		
Necesidad	☒ Esencial	☐ Deseable	☐ Opcional
Prioridad	☒ Alta	☐ Media	☐ Baja
Fuente	☒ Tutor	☐ Alumno	
Verificabilidad	☒ Alta	☐ Media	☐ Baja
Descripción	Con el método de entrada gráfica, el usuario podrá regular en cualquier momento la velocidad a la que se realiza el entrenamiento.		

Tabla 14: requisito funcional 14

ID	RFU-015		
Necesidad	☒ Esencial	☐ Deseable	☐ Opcional
Prioridad	☒ Alta	☐ Media	☐ Baja
Fuente	☒ Tutor	☐ Alumno	
Verificabilidad	☒ Alta	☐ Media	☐ Baja
Descripción	Con el método de entrada gráfica, el usuario tendrá la opción de dejar de utilizarlo para seleccionar método de entrada de nuevo.		

Tabla 15: requisito funcional 15

2.2.1.3. Relacionados con el método de entrada por fichero

ID	RFU-016		
Necesidad	☐ Esencial	☒ Deseable	☐ Opcional
Prioridad	☐ Alta	☒ Media	☐ Baja
Fuente	☒ Tutor	☐ Alumno	
Verificabilidad	☒ Alta	☐ Media	☐ Baja
Descripción	Si se ha seleccionado un archivo de entrenamiento cuyos datos tienen clase, se informará al usuario de que se realizará la calibración del mapa tras el entrenamiento.		

Tabla 16: requisito funcional 16

ID	RFU-017		
Necesidad	☐ Esencial	☒ Deseable	☐ Opcional
Prioridad	☐ Alta	☒ Media	☐ Baja
Fuente	☒ Tutor	☐ Alumno	
Verificabilidad	☒ Alta	☐ Media	☐ Baja
Descripción	Si se muestra la gráfica usando un fichero de dos dimensiones con datos con clase, se permitirá al usuario ver sobre la gráfica a qué clase pertenece cada dato y cada neurona.		

Tabla 17: requisito funcional 17

2.2.1.4. Relacionados con la configuración de la red

ID	RFU-018		
Necesidad	☒ Esencial	☐ Deseable	☐ Opcional
Prioridad	☒ Alta	☐ Media	☐ Baja
Fuente	☒ Tutor	☐ Alumno	
Verificabilidad	☒ Alta	☐ Media	☐ Baja
Descripción	Dos parámetros para la configuración de la red serán la cantidad de filas y de columnas del mapa. Serán dos enteros positivos en el rango [1, N]. No tendrán valores por defecto		

Tabla 18: requisito funcional 18

ID	RFU-019		
Necesidad	☒ Esencial	☐ Deseable	☐ Opcional
Prioridad	☒ Alta	☐ Media	☐ Baja
Fuente	☒ Tutor	☐ Alumno	
Verificabilidad	☒ Alta	☐ Media	☐ Baja
Descripción	Un parámetro de la red será el tipo de vecindario. Podrá ser: • Rectangular. Valor por defecto. • Hexagonal.		

Tabla 19: requisito funcional 19

ID	RFU-020		
Necesidad	☒ Esencial	☐ Deseable	☐ Opcional
Prioridad	☒ Alta	☐ Media	☐ Baja
Fuente	☒ Tutor	☐ Alumno	
Verificabilidad	☒ Alta	☐ Media	☐ Baja
Descripción	Un parámetro de la red será el tipo de distancia de vecindario. Podrá ser: • Euclídea. Valor por defecto. • Pasos. (Cuenta el número mínimo de enlaces entre las dos neuronas cuya distancia se mide).		

Tabla 20: requisito funcional 20

ID	RFU-021		
Necesidad	☒ Esencial	☐ Deseable	☐ Opcional
Prioridad	☒ Alta	☐ Media	☐ Baja
Fuente	☒ Tutor	☐ Alumno	
Verificabilidad	☒ Alta	☐ Media	☐ Baja
Descripción	Un parámetro de la red será el tipo de decrecimiento del vecindario. Podrá ser: • Lineal. • Exponencial. Valor por defecto.		

Tabla 21: requisito funcional 21

ID	RFU-022		
Necesidad	☒ Esencial	☐ Deseable	☐ Opcional
Prioridad	☒ Alta	☐ Media	☐ Baja
Fuente	☒ Tutor	☐ Alumno	
Verificabilidad	☒ Alta	☐ Media	☐ Baja
Descripción	Un parámetro de la red será el límite de vecindario inicial. Se trata de un entero no negativo cuyo valor por defecto será 0.		

Tabla 22: requisito funcional 22

ID	RFU-023		
Necesidad	☒ Esencial	☐ Deseable	☐ Opcional
Prioridad	☒ Alta	☐ Media	☐ Baja
Fuente	☒ Tutor	☐ Alumno	
Verificabilidad	☒ Alta	☐ Media	☐ Baja
Descripción	Un parámetro de la red será la opción de disminuir (o no) durante el entrenamiento el límite de vecindario elegido inicialmente. Por defecto si se disminuirá.		

Tabla 23: requisito funcional 23

ID	RFU-024		
Necesidad	☒ Esencial	☐ Deseable	☐ Opcional
Prioridad	☒ Alta	☐ Media	☐ Baja
Fuente	☒ Tutor	☐ Alumno	
Verificabilidad	☒ Alta	☐ Media	☐ Baja
Descripción	Dos parámetros de la red serán dos números reales entre los que inicializar aleatoriamente los pesos. Sus valores por defecto serán 0 y 1.		

Tabla 24: requisito funcional 24

ID	RFU-025		
Necesidad	☒ Esencial	☐ Deseable	☐ Opcional
Prioridad	☒ Alta	☐ Media	☐ Baja
Fuente	☒ Tutor	☐ Alumno	
Verificabilidad	☒ Alta	☐ Media	☐ Baja
Descripción	Un parámetro de la red será un número real en el rango [0,1] para indicar la tasa de aprendizaje inicial.		

Tabla 25: requisito funcional 25

ID	RFU-026		
Necesidad	☒ Esencial	☐ Deseable	☐ Opcional
Prioridad	☒ Alta	☐ Media	☐ Baja
Fuente	☒ Tutor	☐ Alumno	
Verificabilidad	☒ Alta	☐ Media	☐ Baja
Descripción	Un parámetro de la red será el tipo de decrecimiento de la tasa de aprendizaje. Podrá tomar dos valores: • Lineal. • Exponencial. Valor por defecto.		

Tabla 26: requisito funcional 26

ID	RFU-027		
Necesidad	☒ Esencial	☐ Deseable	☐ Opcional
Prioridad	☒ Alta	☐ Media	☐ Baja
Fuente	☒ Tutor	☐ Alumno	
Verificabilidad	☒ Alta	☐ Media	☐ Baja
Descripción	Un parámetro de la red será la cantidad de ciclos que se entrenará la red. Se trata de un entero mayor que 0 cuyo valor por defecto será 1000.		

Tabla 27: requisito funcional 27

ID	RFU-028		
Necesidad	☒ Esencial	☐ Deseable	☐ Opcional
Prioridad	☒ Alta	☐ Media	☐ Baja
Fuente	☒ Tutor	☐ Alumno	
Verificabilidad	☒ Alta	☐ Media	☐ Baja
Descripción	Un parámetro de la red será una opción para mostrar el entrenamiento gráficamente. Por defecto estará desactivada y sólo estará activada cuando esté seleccionado para entrenamiento un archivo con dos atributos para los datos.		

Tabla 28: requisito funcional 28

2.2.1.5. Relacionados con el proceso de entrenamiento

ID	RFU-029		
Necesidad	☒ Esencial	☐ Deseable	☐ Opcional
Prioridad	☒ Alta	☐ Media	☐ Baja
Fuente	☒ Tutor	☐ Alumno	
Verificabilidad	☒ Alta	☐ Media	☐ Baja
Descripción	La aplicación realizará el entrenamiento de acuerdo a los parámetros proporcionados por el usuario.		

Tabla 29: requisito funcional 29

ID	RFU-030		
Necesidad	☐ Esencial	☒ Deseable	☐ Opcional
Prioridad	☐ Alta	☒ Media	☐ Baja
Fuente	☒ Tutor	☐ Alumno	
Verificabilidad	☒ Alta	☐ Media	☐ Baja
Descripción	Si el conjunto de datos de entrenamiento tiene clase, la aplicación realizará la calibración del mapa como fase final del entrenamiento.		

Tabla 30: requisito funcional 30

ID	RFU-031		
Necesidad	☒ Esencial	☐ Deseable	☐ Opcional
Prioridad	☒ Alta	☐ Media	☐ Baja
Fuente	☒ Tutor	☐ Alumno	
Verificabilidad	☒ Alta	☐ Media	☐ Baja
Descripción	El usuario podrá cancelar el entrenamiento en cualquier momento de su transcurso.		

Tabla 31: requisito funcional 31

ID	RFU-032		
Necesidad	☐ Esencial	☒ Deseable	☐ Opcional
Prioridad	☐ Alta	☒ Media	☐ Baja
Fuente	☒ Tutor	☐ Alumno	
Verificabilidad	☒ Alta	☐ Media	☐ Baja
Descripción	El usuario podrá pausar el entrenamiento en cualquier momento de su transcurso.		

Tabla 32: requisito funcional 32

ID	RFU-033		
Necesidad	☐ Esencial	☒ Deseable	☐ Opcional
Prioridad	☐ Alta	☒ Media	☐ Baja
Fuente	☒ Tutor	☐ Alumno	
Verificabilidad	☒ Alta	☐ Media	☐ Baja
Descripción	El usuario podrá reanudar un entrenamiento pausado.		

Tabla 33: requisito funcional 33

ID	RFU-034		
Necesidad	☐ Esencial	☒ Deseable	☐ Opcional
Prioridad	☐ Alta	☒ Media	☐ Baja
Fuente	☒ Tutor	☐ Alumno	
Verificabilidad	☒ Alta	☐ Media	☐ Baja
Descripción	El usuario podrá inicializar el conjunto de pesos de la red antes de entrenarla, en base a los valores de los parámetros para tal fin. Si lo hace, la aplicación comenzará el entrenamiento partiendo de esos pesos. En caso contrario, la aplicación inicializará los pesos de acuerdo a los parámetros especificados por el usuario para ello.		

Tabla 34: requisito funcional 34

2.2.1.6. Relacionados con la salida

ID	RFU-035		
Necesidad	☒ Esencial	☐ Deseable	☐ Opcional
Prioridad	☒ Alta	☐ Media	☐ Baja
Fuente	☒ Tutor	☐ Alumno	
Verificabilidad	☒ Alta	☐ Media	☐ Baja
Descripción	Con el modo de entrada gráfica, el usuario podrá guardar los datos contenidos en la misma en un fichero, cuyo formato será el mismo que el de los ficheros de entrada.		

Tabla 35: requisito funcional 35

ID	RFU-036		
Necesidad	☒ Esencial	☐ Deseable	☐ Opcional
Prioridad	☒ Alta	☐ Media	☐ Baja
Fuente	☒ Tutor	☐ Alumno	
Verificabilidad	☒ Alta	☐ Media	☐ Baja
Descripción	Tras finalizar un entrenamiento, el usuario podrá almacenar un fichero en el que se indique la posición en la que ha finalizado cada neurona, sus coordenadas en el mapa y sus pesos. También se indicará a qué clase pertenece cada una, en el caso de que los datos de entrenamiento tuvieran clase.		

Tabla 36: requisito funcional 36

ID	RFU-037		
Necesidad	☒ Esencial	☐ Deseable	☐ Opcional
Prioridad	☒ Alta	☐ Media	☐ Baja
Fuente	☒ Tutor	☐ Alumno	
Verificabilidad	☒ Alta	☐ Media	☐ Baja
Descripción	Tras finalizar un entrenamiento, el usuario podrá almacenar un fichero en el que se especifique qué neurona representa a cada dato de entrenamiento y a qué distancia euclídea del mismo se encuentra. Además, si los datos tienen clase, se especificará en qué clase se clasifica cada dato y si está o no bien clasificado. Se incluirá también un porcentaje indicando la tasa de aciertos.		

Tabla 37: requisito funcional 37

ID	RFU-038		
Necesidad	☒ Esencial	☐ Deseable	☐ Opcional
Prioridad	☒ Alta	☐ Media	☐ Baja
Fuente	☒ Tutor	☐ Alumno	
Verificabilidad	☒ Alta	☐ Media	☐ Baja
Descripción	Tras finalizar un entrenamiento, si previo al mismo el usuario seleccionó un fichero de test, podrá almacenar un fichero en el que se especifique qué neurona representa a cada dato y a qué distancia euclídea del mismo se encuentra. Además, si los datos tienen clase, se especificará en qué clase se clasifica cada dato y si está o no bien clasificado. Se incluirá también un porcentaje indicando la tasa de aciertos.		

Tabla 38: requisito funcional 38

ID	RFU-039		
Necesidad	☒ Esencial	☐ Deseable	☐ Opcional
Prioridad	☒ Alta	☐ Media	☐ Baja
Fuente	☒ Tutor	☐ Alumno	
Verificabilidad	☒ Alta	☐ Media	☐ Baja
Descripción	Tras finalizar un entrenamiento, el usuario podrá almacenar un fichero especificando cuál fue el sumatorio de distancias euclídeas entre cada dato y la neurona que lo representaba al final de cada ciclo.		

Tabla 39: requisito funcional 39

ID	RFU-040		
Necesidad	☐ Esencial	☒ Deseable	☐ Opcional
Prioridad	☐ Alta	☒ Media	☐ Baja
Fuente	☒ Tutor	☐ Alumno	
Verificabilidad	☒ Alta	☐ Media	☐ Baja
Descripción	Tras finalizar un entrenamiento, el usuario podrá ver una gráfica con la evolución del sumatorio de distancias euclídeas entre cada dato y la neurona que lo representa a lo largo del proceso de entrenamiento.		

Tabla 40: requisito funcional 40

2.2.1.7. Generales

ID	RFU-041		
Necesidad	☐ Esencial	☒ Deseable	☐ Opcional
Prioridad	☒ Alta	☐ Media	☐ Baja
Fuente	☒ Tutor	☐ Alumno	
Verificabilidad	☐ Alta	☒ Media	☐ Baja
Descripción	La aplicación funcionará en diferentes sistemas operativos. Al menos, en Windows 7, Windows 8 y Ubuntu 12.10.		

Tabla 41: requisito funcional 41

ID	RFU-042		
Necesidad	☐ Esencial	☐ Deseable	☒ Opcional
Prioridad	☐ Alta	☐ Media	☒ Baja
Fuente	☒ Tutor	☐ Alumno	
Verificabilidad	☒ Alta	☐ Media	☐ Baja
Descripción	Cuando la aplicación sea iniciada, mostrará al usuario información indicando que se trata de un Trabajo de Fin de Grado, así como quién es su autor y su director.		

Tabla 42: requisito funcional 42

2.2.2. Requisitos no funcionales

2.2.2.1. Relacionados con el método de entrada por fichero

ID	RNF-001		
Necesidad	☒ Esencial	☐ Deseable	☐ Opcional
Prioridad	☒ Alta	☐ Media	☐ Baja
Fuente	☒ Tutor	☐ Alumno	
Verificabilidad	☒ Alta	☐ Media	☐ Baja

Descripción	El formato de los ficheros de entrada (tanto de entrenamiento como de test) será el siguiente:
	• Una primera línea que contiene únicamente un entero indicando la cantidad de atributos de los datos.
	• Tantas líneas como datos tenga el fichero, con la siguiente estructura:
	  ○ <A1><sep><A2><sep>…<sep>[<Clase>]
	• <A<número>>hace referencia a los distintos atributos, que tendrán valores numéricos reales.
	• sep es el separador, que podrá ser:
	  ○ Una coma.
	  ○ Una sucesión de espacios en blanco (espacios, tabulaciones, etc.) precedidos, o no, por una coma.
	• Si los datos tienen clase, aparecerá al final de la línea, en caso contrario, no aparecerá, terminando la línea con el último atributo de los datos.		

Tabla 43: requisito no funcional 1

2.2.2.2. Relacionados con el método de entrada gráfica

ID	RNF-002		
Necesidad	☒ Esencial	☐ Deseable	☐ Opcional
Prioridad	☒ Alta	☐ Media	☐ Baja
Fuente	☒ Tutor	☐ Alumno	
Verificabilidad	☒ Alta	☐ Media	☐ Baja
Descripción	Cuando el usuario haya elegido como método de entrada la entrada gráfica, la aplicación mostrará una sección rectangular lo más amplia posible (sin exceder los límites de la pantalla) donde se realizará el entrenamiento.		

Tabla 44: requisito no funcional 2

ID	RNF-003		
Necesidad	☒ Esencial	☐ Deseable	☐ Opcional
Prioridad	☒ Alta	☐ Media	☐ Baja
Fuente	☒ Tutor	☐ Alumno	
Verificabilidad	☒ Alta	☐ Media	☐ Baja
Descripción	Cuando el usuario haya elegido como método de entrada la entrada gráfica, la esquina inferior izquierda corresponderá al punto (0,0) y la superior izquierda al punto (0,1). El escalado horizontal será equivalente al vertical.		

Tabla 45: requisito no funcional 3

ID	RNF-004		
Necesidad	☒ Esencial	☐ Deseable	☐ Opcional
Prioridad	☒ Alta	☐ Media	☐ Baja
Fuente	☐ Tutor	☒ Alumno	
Verificabilidad	☒ Alta	☐ Media	☐ Baja
Descripción	Con el método de entrada gráfica, cada dato se representará, por defecto, como un cuadrado negro (RGB (0, 0, 0)) de lado 4 píxeles centrado en la posición del dato que representa.		

Tabla 46: requisito no funcional 4

ID	RNF-005		
Necesidad	☒ Esencial	☐ Deseable	☐ Opcional
Prioridad	☒ Alta	☐ Media	☐ Baja
Fuente	☐ Tutor	☒ Alumno	
Verificabilidad	☒ Alta	☐ Media	☐ Baja
Descripción	Con el método de entrada gráfica, cada neurona se representará, por defecto, como un círculo azul (RGB (0, 0, 255)) de radio 4 píxeles centrado en la posición de la neurona que representa.		

Tabla 47: requisito no funcional 5

ID	RNF-006		
Necesidad	☐ Esencial	☒ Deseable	☐ Opcional
Prioridad	☐ Alta	☒ Media	☐ Baja
Fuente	☐ Tutor	☒ Alumno	
Verificabilidad	☒ Alta	☐ Media	☐ Baja
Descripción	Con el método de entrada gráfica, la forma de inicializar neuronas será con clics con el botón derecho del ratón. Un clic supondrá una neurona añadida en la posición del ratón. Se añadirán si las dimensiones y la topología de la red tienen valores correctos. Sólo será posible si la red no está siendo entrenada.		

Tabla 48: requisito no funcional 6

ID	RNF-007		
Necesidad	☐ Esencial	☒ Deseable	☐ Opcional
Prioridad	☐ Alta	☒ Media	☐ Baja
Fuente	☒ Tutor	☐ Alumno	
Verificabilidad	☒ Alta	☐ Media	☐ Baja
Descripción	Con el método de entrada gráfica, la forma de cambiar la posición de neuronas inicializadas será con un clic con el botón derecho sobre ella y arrastrándola hasta la nueva posición. Sólo se podrá hacer si la red no está siendo entrenada.		

Tabla 49: requisito no funcional 7

ID	RNF-008		
Necesidad	☒ Esencial	☐ Deseable	☐ Opcional
Prioridad	☒ Alta	☐ Media	☐ Baja
Fuente	☒ Tutor	☐ Alumno	
Verificabilidad	☒ Alta	☐ Media	☐ Baja
Descripción	Con el método de entrada gráfica, el fondo se representará de color blanco (RGB (255, 255, 255)) y con el texto del color inverso.		

Tabla 50: requisito no funcional 8

ID	RNF-009		
Necesidad	☐ Esencial	☐ Deseable	☒ Opcional
Prioridad	☐ Alta	☐ Media	☒ Baja
Fuente	☒ Tutor	☐ Alumno	
Verificabilidad	☒ Alta	☐ Media	☐ Baja
Descripción	Con el método de entrada gráfica, el usuario podrá modificar el color con el que se representan las neuronas por el que prefiera.		

Tabla 51: requisito no funcional 9

ID	RNF-010		
Necesidad	☐ Esencial	☐ Deseable	☒ Opcional
Prioridad	☐ Alta	☐ Media	☒ Baja
Fuente	☒ Tutor	☐ Alumno	
Verificabilidad	☒ Alta	☐ Media	☐ Baja
Descripción	Con el método de entrada gráfica, el usuario podrá modificar el color del fondo por el que prefiera.		

Tabla 52: requisito no funcional 10

ID	RNF-011		
Necesidad	☐ Esencial	☐ Deseable	☒ Opcional
Prioridad	☐ Alta	☐ Media	☒ Baja
Fuente	☒ Tutor	☐ Alumno	
Verificabilidad	☒ Alta	☐ Media	☐ Baja
Descripción	Con el método de entrada gráfica, el usuario podrá modificar el color con el que se representan los datos por el que prefiera.		

Tabla 53: requisito no funcional 11

2.2.2.3. Generales

ID	RNF-012		
Necesidad	☐ Esencial	☐ Deseable	☒ Opcional
Prioridad	☐ Alta	☐ Media	☒ Baja
Fuente	☒ Tutor	☐ Alumno	
Verificabilidad	☒ Alta	☐ Media	☐ Baja
Descripción	Se utilizará el punto (.) como separador entre la parte entera y la decimal de todos los números.		

Tabla 54: requisito no funcional 12

2.2.3. Requisitos negativos

ID	RNE-001		
Necesidad	☒ Esencial	☐ Deseable	☐ Opcional
Prioridad	☒ Alta	☐ Media	☐ Baja
Fuente	☒ Tutor	☐ Alumno	
Verificabilidad	☒ Alta	☐ Media	☐ Baja
Descripción	El usuario no podrá seleccionar ficheros, ni cambiar el tipo de entrada, ni modificar ningún parámetro de la red mientras está siendo entrenada.		

Tabla 55: requisito negativo 1

ID	RNE-002		
Necesidad	☒ Esencial	☐ Deseable	☐ Opcional
Prioridad	☒ Alta	☐ Media	☐ Baja
Fuente	☒ Tutor	☐ Alumno	
Verificabilidad	☒ Alta	☐ Media	☐ Baja
Descripción	El usuario no podrá modificar ni las dimensiones de la red ni el tipo de vecindario mientras se esté mostrando en la gráfica.		

Tabla 56: requisito negativo 2

2.3. Marco regulador

En este apartado, se explica cuál es el marco regulador legal que afecta a la aplicación, tanto a los usuarios que la utilicen como en el desarrollo de la misma.

La aplicación desarrollada puede leer ficheros de datos, cuya obtención/generación y uso deben estar siempre regidos por la *Ley Orgánica 15/1999, de 13 de diciembre, de Protección de Datos de Carácter Personal*. Deben cumplir esta ley tanto los ficheros que se han usado para desarrollo y pruebas sobre la aplicación como los utilizados por usuarios. [19]

Respecto al software utilizado para el desarrollo de la aplicación, hay que tener en cuenta que al adquirirlo y utilizarlo se acepta un acuerdo de licencia que se debe cumplir. Se han respetado las licencias de:

- Eclipse [20]
- Microsoft Windows 8 [21]
- Microsoft Office 2013 [22]
- JMathPlot [16]

Por último, es importante destacar que se debe respetar el *Real Decreto Legislativo 1/1996, de 12 de abril, por el que se aprueba el texto refundido de la Ley de Propiedad Intelectual, regularizando, aclarando y armonizando las disposiciones legales vigentes sobre la materia*. En este trabajo, se ha adquirido conocimiento y se han expuesto datos de diferentes fuentes, las cuales son referenciadas en el apartado “8. Referencias”. [23]

3. Diseño de la solución técnica

3.1. Alternativas de diseño

Para el diseño del programa, es importante tener en cuenta que el lenguaje de programación elegido es Java, cuyo paradigma de programación es la orientación a objetos.

Para la disposición de los elementos visuales en la interfaz de usuario, Java proporciona diferentes plantillas o *layouts*. Se ha optado por usar "GridBagLayout" que, tras compararla con las demás en la web de Oracle, parece que es una de las más flexibles para organizar los elementos como se desee y para redimensionar la aplicación especificando su comportamiento al hacerlo. Además, como desde el principio no se cuenta con una idea clara y exacta de cómo debe ser la aplicación gráficamente (mientras que ofrezca las opciones necesarias), se ha optado por utilizar uno de los *layout* más tolerantes a posibles cambios. [11][12]

Permite organizar los elementos como si el espacio disponible fuera una tabla con la cantidad de filas y columnas elegidas y en sus celdas se colocan los elementos. Además, todas las filas no tienen por qué tener el mismo alto ni las columnas el mismo ancho y es posible hacer que un elemento ocupe varias filas y/o columnas si se desea. Este tipo de organización permite redimensionar la ventana de la aplicación por parte del usuario y que al hacerlo sus elementos se redimensionen también si así se ha establecido. [13]

Para la parte gráfica, se ha decidido usar las librerías "Graphics" y "Graphics2D" de Java. Entre las dos proporcionan toda la funcionalidad básica necesaria para dibujar cada ciclo del proceso de entrenamiento de la red, con un uso bastante sencillo para lo requerido en la aplicación. No se ha considerado necesario contemplar el uso de otras librerías, ya que la aplicación mostrará formas sencillas, que fácilmente pueden obtenerse con las librerías de Java mencionadas y cuyo uso puede consultarse en la API (Application Programming Interface) y en otras webs de Oracle que enseñan cómo hacer algunas cosas en Java. [15]

En el caso de que la parte gráfica hubiera sido más compleja, por ejemplo, usando muchas formas geométricas diferentes, ejes cartesianos, imágenes y/o animaciones, figuras en tres dimensiones, permitir al usuario cambiar el punto de vista, etc. sí que se habría estudiado la posibilidad de utilizar otras librerías.

Para la parte de la gráfica que muestra la evolución del error a lo largo del entrenamiento, se ha optado por utilizar la librería "JMathPlot", [16] que permite hacerlo de manera sencilla. Existen alternativas como "JFreeChart", [17] con más opciones, pero en este caso no se necesita hacer gráficas más complejas, por lo que es suficiente utilizar "JMathPlot", ahorrando espacio en el tamaño final de la aplicación sin perder funcionalidad.

El entrenamiento gráfico es una parte esencial de la aplicación. Por ello, es importante garantizar que se muestra de forma correcta en todo momento. Puesto que la aplicación tiene que mostrarse continuamente en la pantalla y realizar los cálculos del proceso de entrenamiento, es necesario utilizar hilos, para asegurar que no se producen

retrasos en la interfaz mientras se realiza el entrenamiento. Además, debe existir comunicación entre los hilos para tener la certeza de que todo lo que se pinta en la pantalla corresponde a un ciclo concreto del proceso. De lo contrario, podría darse una situación en la que lo mostrado en pantalla no sea coherente. Podría ocurrir, por ejemplo, que se muestre un número de ciclo mientras que la posición de algunas neuronas y la tasa de aprendizaje mostradas se corresponden con el ciclo siguiente.

Para evitar este tipo de situaciones, se ha utilizado un *mutex*, que permite una coordinación adecuada entre las acciones de los hilos. Se trata de un mecanismo de exclusión mutua (MUTual EXclusion) que impide que un hilo ejecute su sección crítica si otro hilo está ya ejecutando la suya y cuando este último termine permitirá que continúe la ejecución del primero. Además, se ha utilizado para exigir que la gráfica sea actualizada al menos una vez por ciclo, independientemente de la gestión de procesos que realice el sistema operativo, de forma que pueda evitarse que se produzcan saltos indeseados.

La gestión de hilos también podría haberse realizado con algún otro mecanismo, como los semáforos. Se utilizó un *mutex* por ser una gestión de procesos sencilla: un proceso entrenaba mientras el otro dibujaba en la pantalla. En caso de que hubiese habido más procesos involucrados utilizando los mismos datos, probablemente habrían sido más adecuados los semáforos, ya que permiten la ejecución a varios procesos a la vez y el hecho de que uno esté ejecutando no excluye necesariamente a todos los demás.

3.2. Diseño de la aplicación

A continuación, se muestra el diagrama de clases de la aplicación, mostrando únicamente las clases y las relaciones entre ellas, sin mostrar atributos ni operaciones:

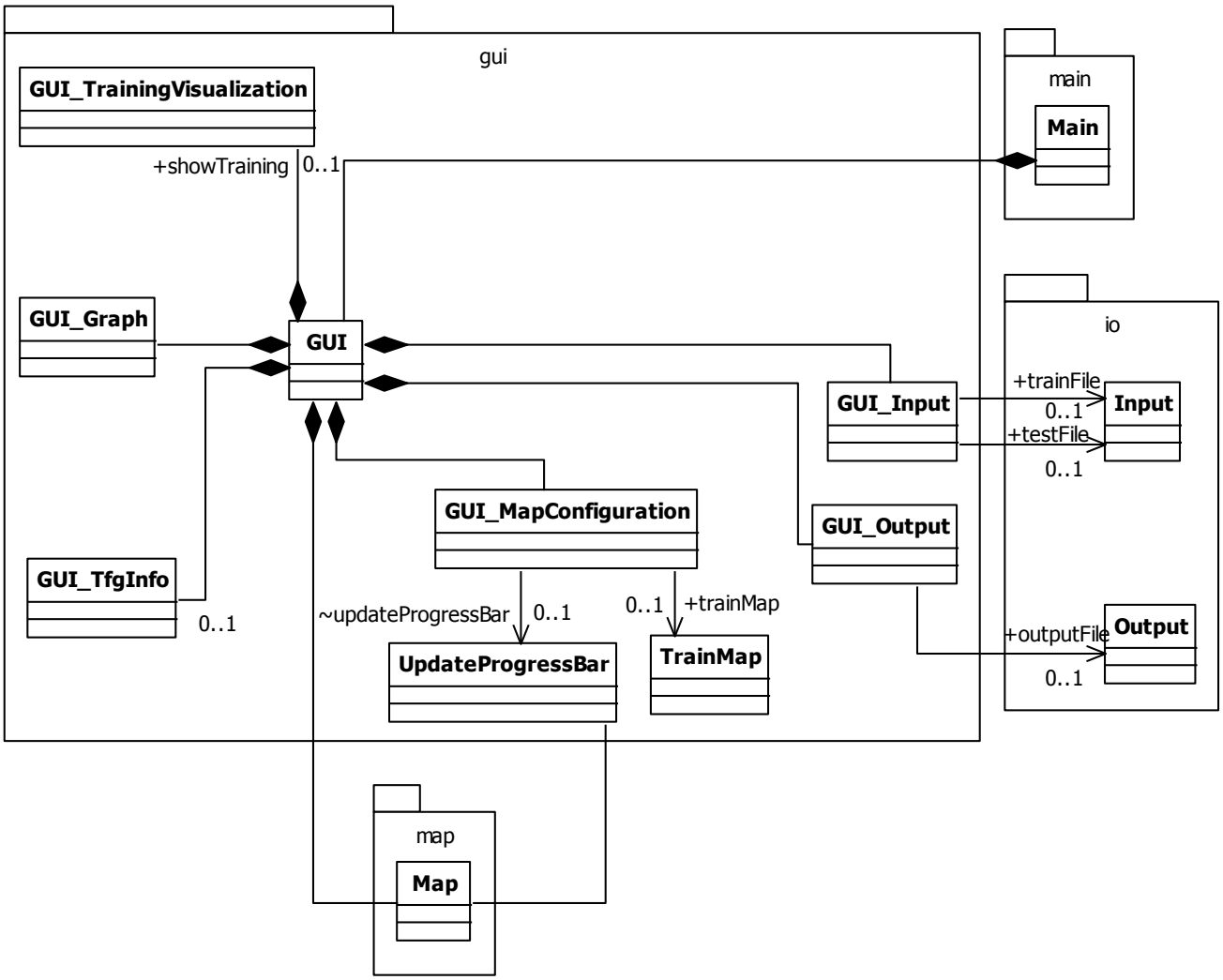


Ilustración 4: diagrama de clases en UML (Unified Modeling Language), excluyendo atributos y operaciones, hecho con WhiteStarUML

En el anexo "8.1. Diagrama de clases por partes" se adjuntan diagramas UML por partes para una mejor visualización.

El código de la aplicación está organizado en varios paquetes:

- main. Contiene la clase "Main". Su único cometido es iniciar la aplicación creando un objeto de la clase "GUI".

- gui. Contiene todas las clases relacionadas con la GUI (Graphic User Interface). Permiten la correcta comunicación entre el usuario y la aplicación. La aplicación, desde un punto de vista visual, se divide en cinco partes:
  - Entrada. La clase encargada de mostrar esta parte es "GUI_Input". Contiene todos los elementos que sirven al usuario para especificar, en caso de haberlos, cuáles son los ficheros de entrenamiento y test o si crear la entrada gráficamente usando el ratón.
  - Configuración del mapa. La clase "GUI_MapConfiguration" es la que contiene todos los elementos que permiten al usuario definir los parámetros del mapa.

- Salida. Sus elementos son mostrados por la clase "GUI_Output". Se trata de un panel que posibilita al usuario guardar diferentes ficheros con información sobre los resultados del entrenamiento de la red. En caso de que haya generado un conjunto de datos gráficamente, también se le permite guardarlo.
- Visualización gráfica del entrenamiento. Las clases "GUI_TrainingVisualization" y "GUI_Graph" son las encargadas de mostrar cómo se realiza el entrenamiento. La clase "GUI_TrainingVisualization" es la que permite borrar los datos y las neuronas, cambiar sus colores, regular la velocidad del entrenamiento y permitir quitar la gráfica para usar ficheros. La clase "GUI_Graph" es la que se encarga de mostrar gráficamente el entrenamiento. El contenido de estas clases no es visible para el usuario en todo momento (si el usuario no usa gráfica para entrenar ve una interfaz más reducida, suficiente para seleccionar los ficheros de entrada y configurar la red).
- Información TFG. Al inicio de la aplicación, aparece una segunda ventana delante de la principal, mostrada por la clase "GUI_TfgInfo". Se trata de una ventana meramente informativa, que hace saber al usuario que la aplicación ha sido desarrollada en una universidad como Trabajo de Fin de Grado. Esta ventana sólo es visible al principio.

La clase GUI contiene la ventana principal donde se integran todos estos elementos. Además, contiene el mapa que será entrenado.

- io. Contiene las clases relacionadas con operaciones de entrada y salida. La clase "Input" se encarga de leer los ficheros de entrada, tanto de entrenamiento como de test, y la clase "Output" se encarga de escribir los ficheros de salida.

- map. Únicamente contiene la clase "Map", que tiene todos los atributos y operaciones necesarios para la definición de un mapa, así como su entrenamiento y su calibración.

4. Resultados y evaluación

En este apartado, se definen una serie de pruebas realizadas sobre la aplicación y sus resultados. Se trata de pruebas creadas desde el punto de vista del usuario, teniendo en cuenta qué cosas son las que le puede interesar hacer y cómo debe proceder para ello.

Dada la utilidad de la aplicación, los siguientes casos de uso (o procedimientos, ya que los casos de uso deberían ser más breves y con una descripción más formal y concreta) son los que se consideran más interesantes para un usuario:

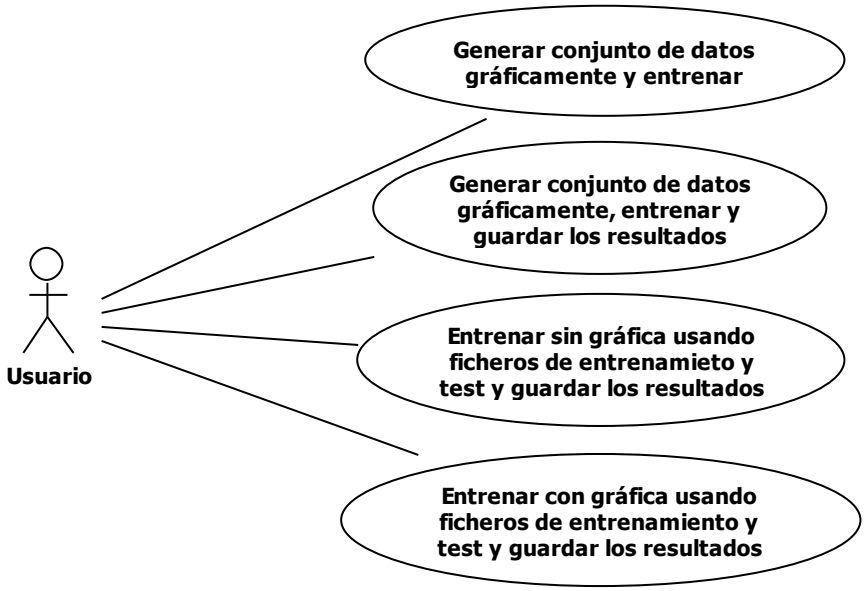


Ilustración 5: diagrama de casos de uso, hecho con WhiteStarUML

Se considera que, por lo general, esto es todo lo que puede querer hacer un usuario con la aplicación.

- Generar conjunto de datos gráficamente y entrenar. Esto puede ser interesante para el usuario que está utilizando la aplicación con el fin de entender de una manera lo más intuitiva posible cómo funcionan los mapas de Kohonen. Los pasos que tiene que realizar el usuario para ello son:
  1. Pulsar sobre el botón "Crear entrada gráficamente".
  2. Hacer clic con el botón izquierdo del ratón sobre la gráfica que ha aparecido en los lugares donde se quieran añadir datos.
  3. Configurar los parámetros del mapa.
  4. Pulsar sobre el botón "Entrenar".

- Generar conjunto de datos gráficamente, entrenar y guardar los resultados. Útil para el usuario que quiera aprender intuitivamente cómo funcionan los mapas de Kohonen y ayudarse para ello de datos más concretos que la visualización del entrenamiento. Tendría que realizar los mismos pasos que en el caso anterior, pero con algunos más al final:
  1. Pulsar sobre el botón "Crear entrada gráficamente".

2. Hacer clic con el botón izquierdo del ratón sobre la gráfica que ha aparecido en los lugares donde se quieran añadir datos.
3. Configurar los parámetros del mapa.
4. Pulsar sobre el botón “Entrenar”.
5. Según los resultados que se quieran guardar, pulsar sobre el botón apropiado para ello.
6. Elegir un destino y un nombre para el archivo que se creará.
7. Volver al paso 5 si se quieren guardar más resultados.

- Entrenar sin gráfica usando ficheros de entrenamiento y test y guardar los resultados. Se trata de una opción útil, sobre todo, para aquellos que ya han entendido la utilidad de los mapas de Kohonen y tienen ficheros de datos sobre los que quieren aplicarlos. Los pasos a seguir son los siguientes:
  1. Pulsar sobre el botón “Seleccionar archivo de entrenamiento”.
  2. Usar el explorador que se ha abierto para seleccionar el archivo de entrenamiento.
  3. Pulsar sobre el botón “Seleccionar archivo de test”.
  4. Usar el explorador para seleccionar el archivo de test.
  5. Configurar los parámetros de la red.
  6. Pulsar sobre el botón “Entrenar”
  7. Según los resultados que se quieran guardar, pulsar sobre el botón apropiado para ello.
  8. Elegir un destino y un nombre para el archivo que se creará.
  9. Volver al paso 7 si se quieren guardar más resultados.

- Entrenar con gráfica usando ficheros de entrenamiento y test y guardar los resultados. Esta opción es interesante para aquellos usuarios que quieren aprender sobre el funcionamiento de los mapas de Kohonen a partir de datos concretos, sin perder por ello la posibilidad de visualizar cómo se produce el entrenamiento. Los pasos son los siguientes:
  1. Pulsar sobre el botón “Seleccionar archivo de entrenamiento”.
  2. Usar el explorador que se ha abierto para seleccionar el archivo de entrenamiento.
  3. Pulsar sobre el botón “Seleccionar archivo de test”.
  4. Usar el explorador para seleccionar el archivo de test.
  5. Pulsar sobre la casilla de “Mostrar entrenamiento gráficamente”.
  6. Configurar los parámetros de la red.
  7. Pulsar sobre el botón “Entrenar”.
  8. Según los resultados que se quieran guardar, pulsar sobre el botón apropiado para ello.
  9. Elegir un destino y un nombre para el archivo que se creará.
  10. Volver al paso 8 si se quieren guardar más resultados.

Cabe decir que, además de seguir estrictamente esos pasos, algunos de ellos pueden cambiarse de orden y existen más opciones para el usuario que podrían haber sido pasos intermedios, como inicializar las neuronas manualmente, cambiar los colores de los datos, neuronas y/o fondo, borrar los datos y/o neuronas de la gráfica, regular la velocidad del entrenamiento, mostrar/ocultar coordenadas de las neuronas, etc.

Además de haberse realizado pruebas tras finalizar la implementación evaluando los casos descritos anteriormente, para garantizar el buen funcionamiento de la aplicación, durante su desarrollo y tras finalizar el mismo se hicieron muchas más pruebas, más exhaustivas que las anteriores y que comprobaban cosas más minuciosas, las cuales han permitido asegurarse de que se cumplía con los requisitos establecidos.

Convenía que algunas de esas pruebas se realizaran justo después de la implementación del código asociado, ya que carecía de sentido avanzar más en el proyecto sin tener cierta seguridad de que algo básico funcionaba correctamente. Más tarde se podría probar más exhaustivamente si fuera necesario. Por el contrario, otras características de la aplicación podían implementarse y no probarse inmediatamente, pudiendo dar más prioridad a otra tarea si era necesario.

Finalmente, se puede afirmar que los resultados de todas las pruebas realizadas han sido satisfactorios, garantizando, en gran medida, la calidad de la aplicación. Una de las consecuencias de que las pruebas hayan ido bien es que se haya llegado a implementar las funcionalidades relacionadas con los requisitos cuya "Necesidad" era "Opcional". Aunque se trate de funcionalidades prescindibles, sí que es cierto que su presencia implica que la aplicación sea más completa y, por tanto, mejor.

A continuación, se mostrarán detalladamente algunos ejemplos de uso de la aplicación, mostrando algunas de sus características principales.

4.1. Ejemplos

4.1.1. Ejemplo 1: distribución de datos uniforme

Con una distribución uniforme aleatoria de los datos:

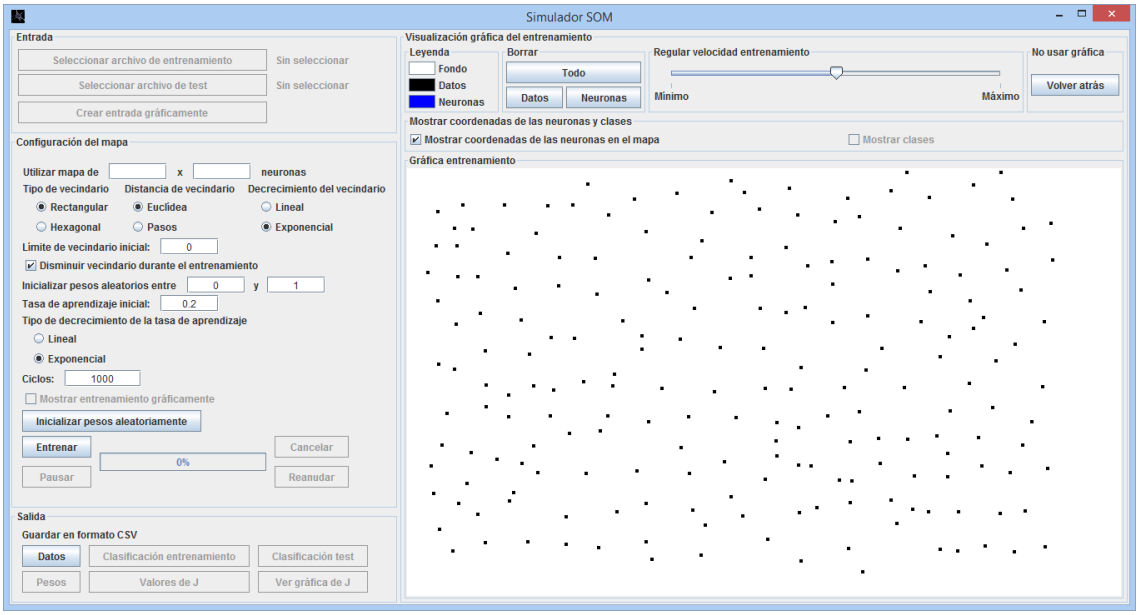


Ilustración 6: ejemplo 1: disposición de los datos

Se ven los datos configurados de forma uniforme y aleatoria, añadidos con clics de ratón.

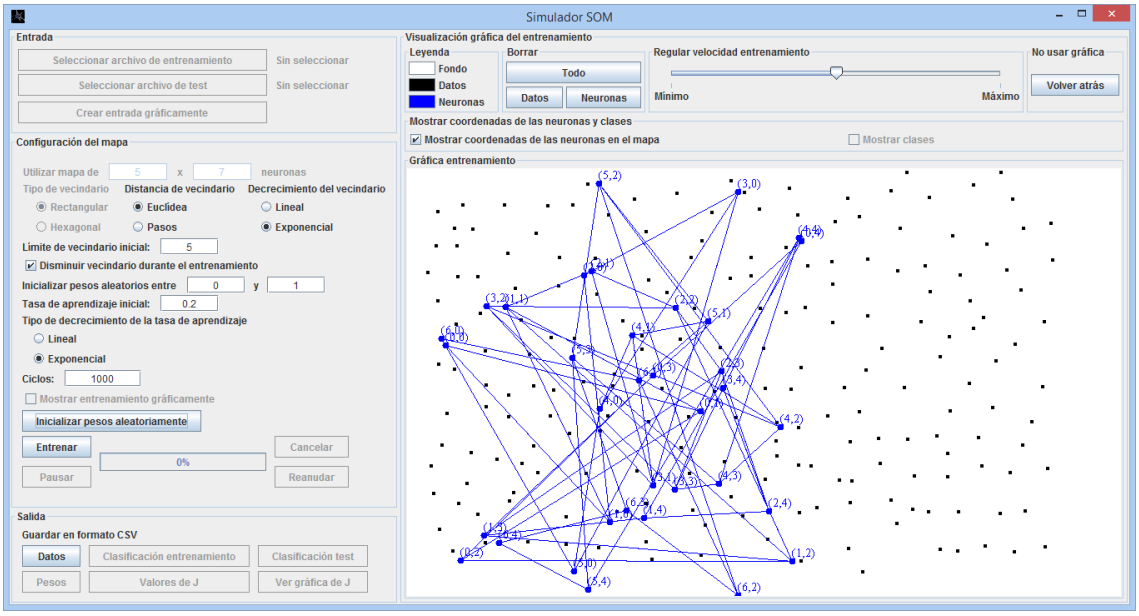


Ilustración 7: ejemplo 1: inicialización aleatoria

Se configura un mapa de 5×7 y se inicializa aleatoriamente. El resto de parámetros son los que tiene la aplicación por defecto, salvo el vecindario que se ha aumentado.

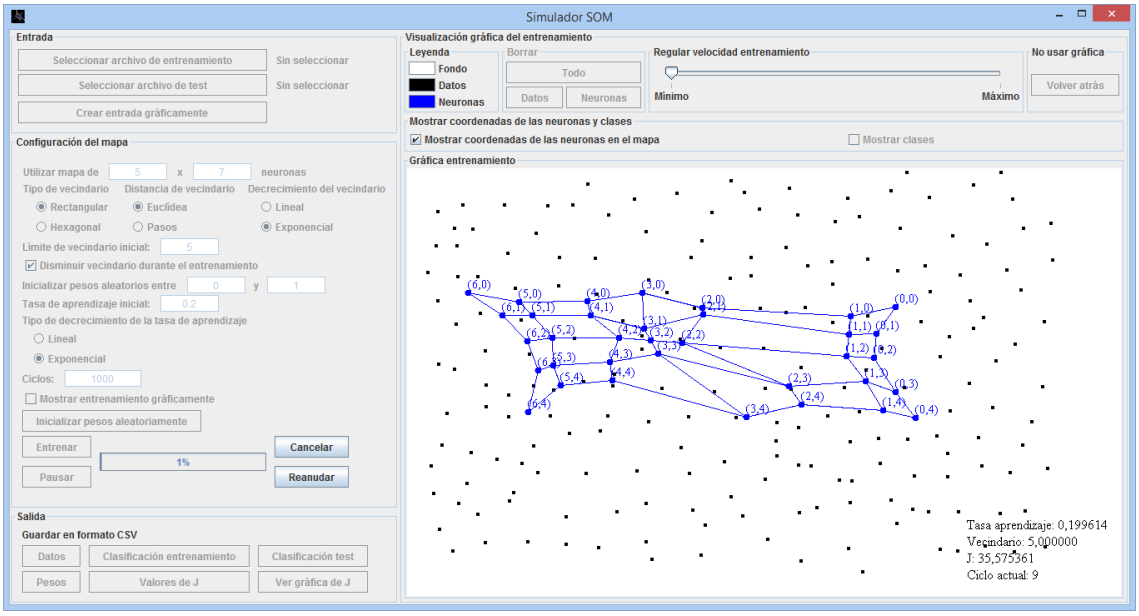


Ilustración 8: ejemplo 1: ciclo 9

Se puede ver como en el ciclo noveno (de un total de mil) el mapa ha cambiado mucho respecto a la inicialización aleatoria. Esto ha ocurrido porque en los primeros ciclos es cuando más altos son el límite de vecindario y la tasa de aprendizaje, permitiendo que se produzcan los cambios más bruscos. La posición del mapa en su conjunto está cerca de la posición central del conjunto de datos.

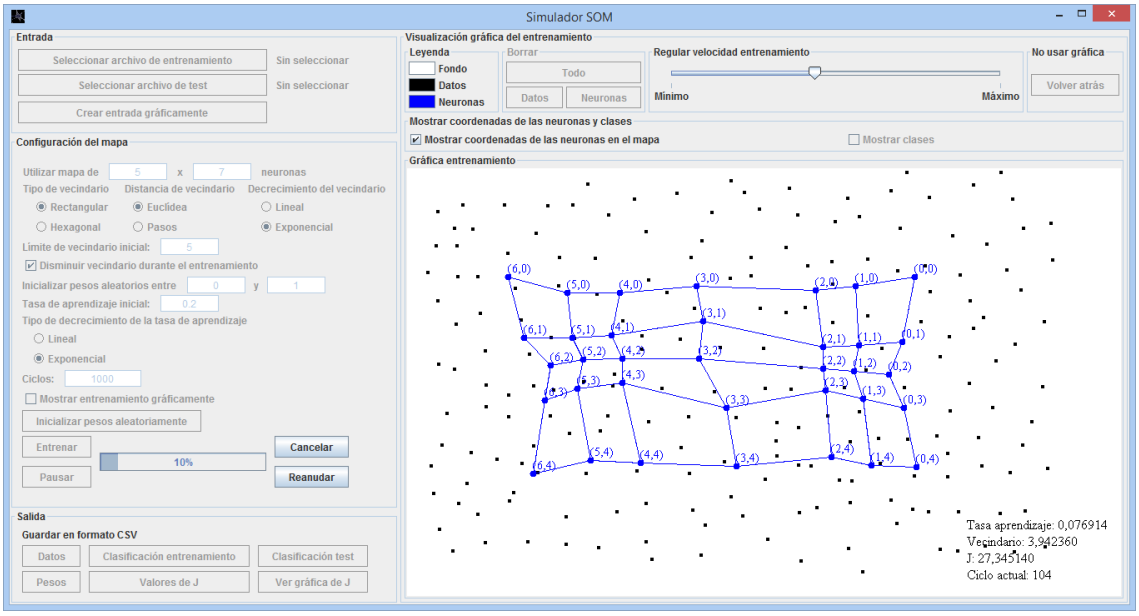


Ilustración 9: ejemplo 1: ciclo 104

En el ciclo 104, el vecindario ha ido disminuyendo y las neuronas se han separado más entre sí.

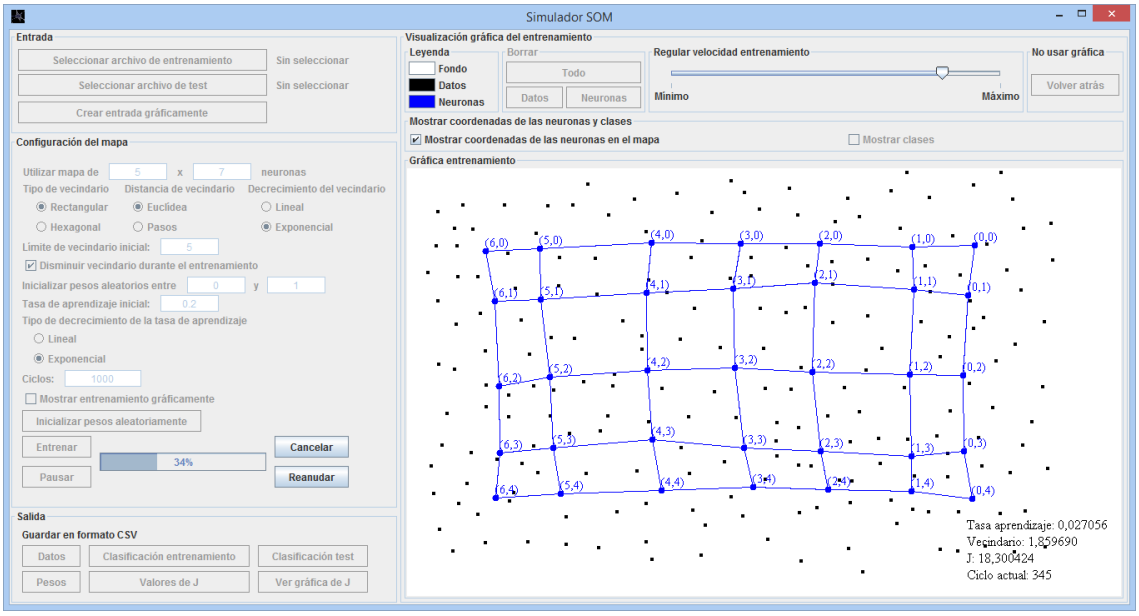


Ilustración 10: ejemplo 1: ciclo 345

Ciclo 345: las neuronas se han separado aún más y la distribución de las mismas en el espacio de entrada es bastante uniforme.

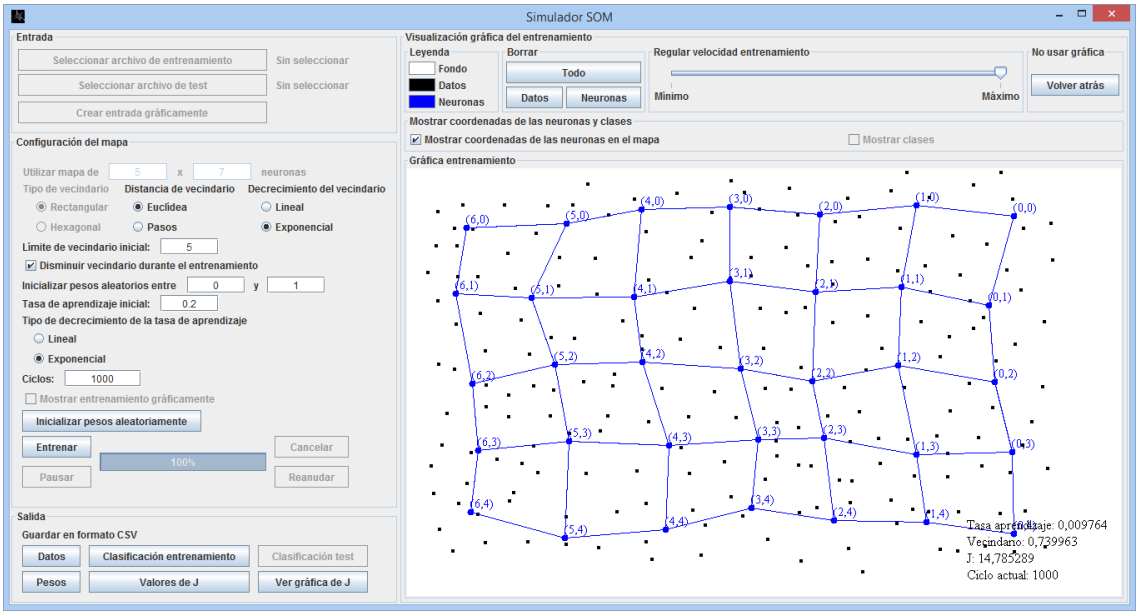


Ilustración 11: ejemplo 1: entrenamiento finalizado

Entrenamiento finalizado: las neuronas han terminado de ajustarse. Se puede observar que, al haber usado una distribución uniforme, las neuronas forman una especie de cuadrícula.

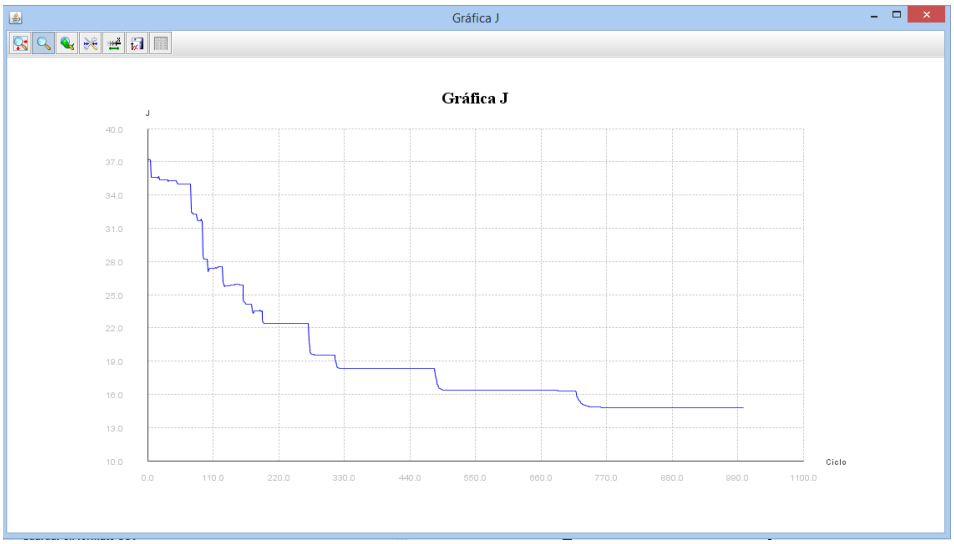


Ilustración 12: ejemplo 1: evolución del error

Evolución del error: es posible ver como el error cometido por la red ha ido descendiendo durante el entrenamiento. Se observan algunos picos, que probablemente denotan momentos en los que la disminución del vecindario implicaba una disminución en la cantidad de neuronas cuyos pesos serían actualizados junto con el de la ganadora.

La medida de error que utiliza la aplicación, a la que denomina J, es la suma de distancias de cada patrón a la neurona más cercana, la que lo representa.

4.1.2. Ejemplo 2: distribución de datos circular

Se expone un ejemplo de uso en el que los datos tienen una distribución más o menos circular:

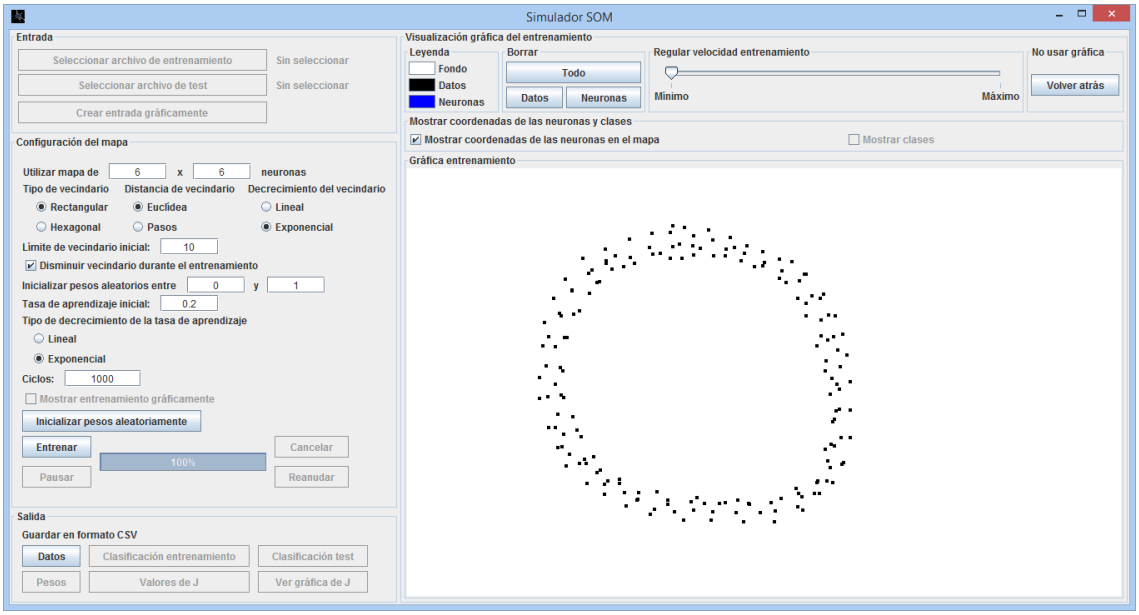


Ilustración 13: ejemplo 2: distribución de los datos

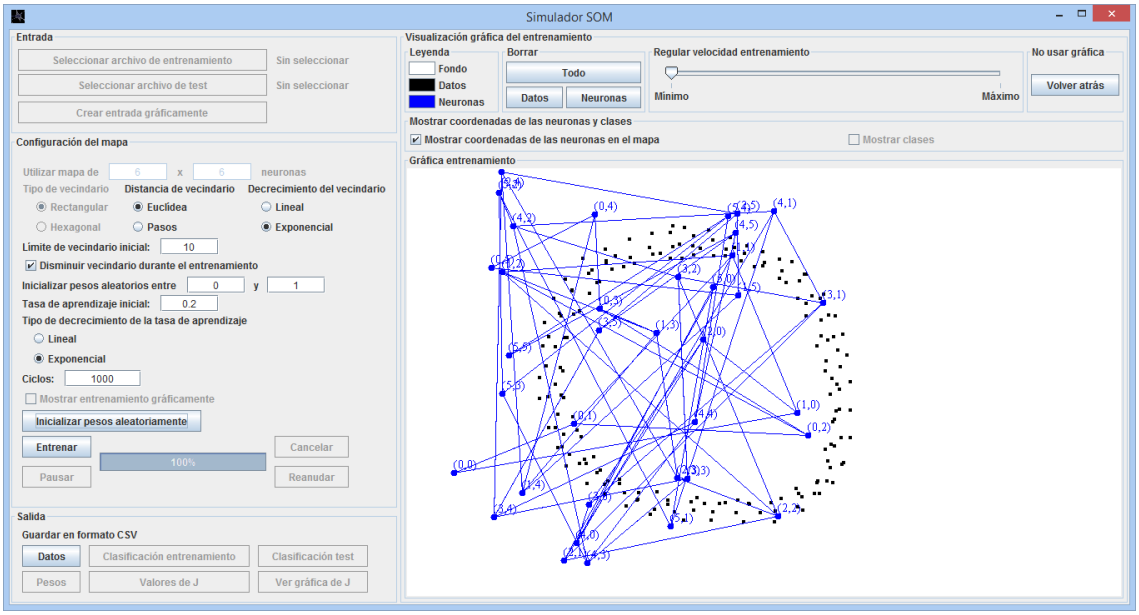


Ilustración 14: ejemplo 2: inicialización aleatoria

Se utiliza un mapa de 6×6 que se inicializa aleatoriamente. El límite de vecindario inicial se ha establecido en 10.

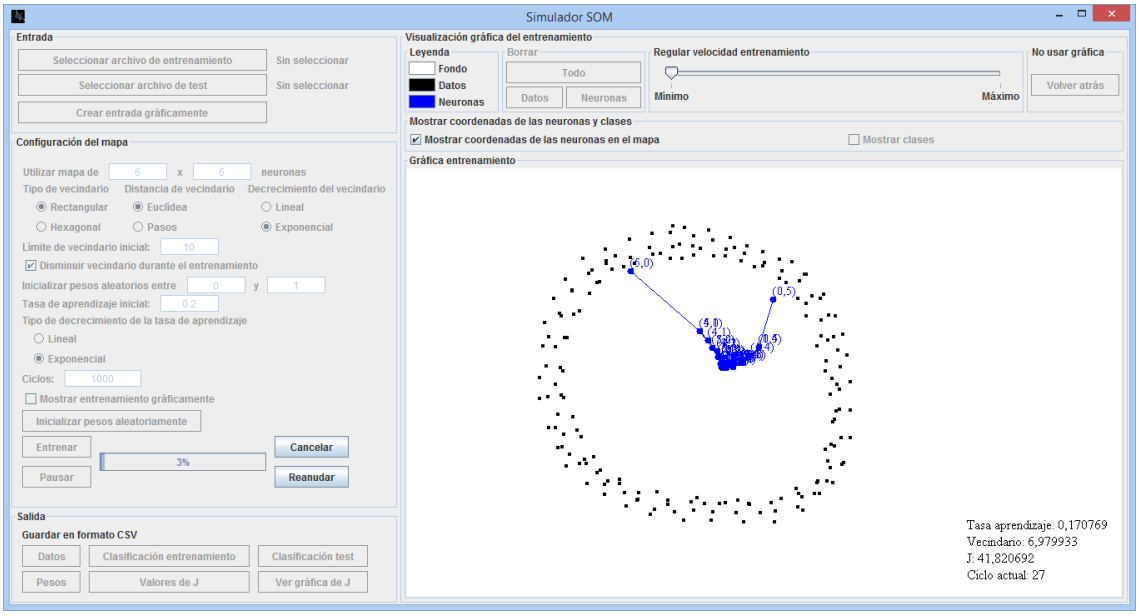


Ilustración 15: ejemplo 2: ciclo 27

Ciclo 27: de nuevo, al comenzar, el mapa ha sufrido un gran cambio respecto a la inicialización aleatoria. El conjunto de neuronas, en general, está cercano al centro del conjunto de datos.

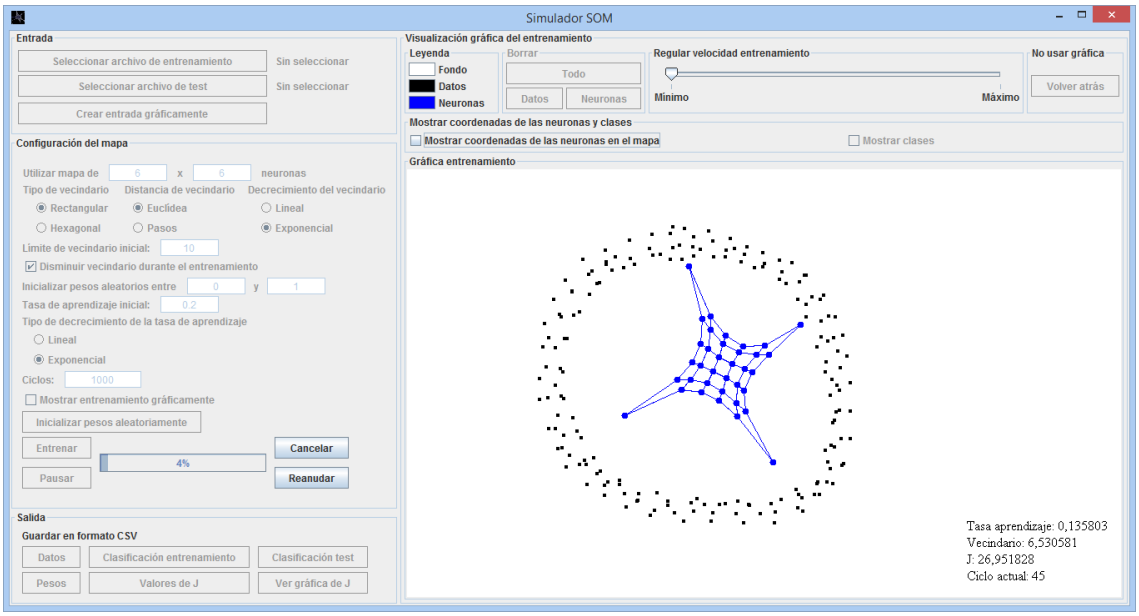


Ilustración 16: ejemplo 2: ciclo 45

Ciclo 45: el conjunto sigue estando cercano al centro, pero esta vez las neuronas están más separadas entre sí.

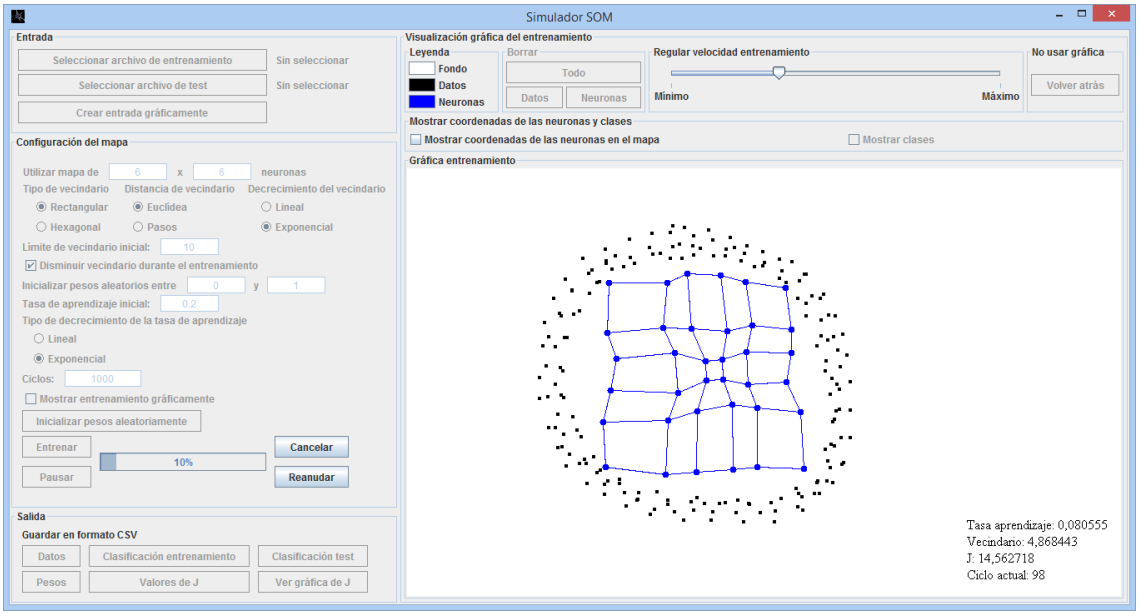


Ilustración 17: ejemplo 2: ciclo 98

Ciclo 98: las neuronas se han separado aún más.

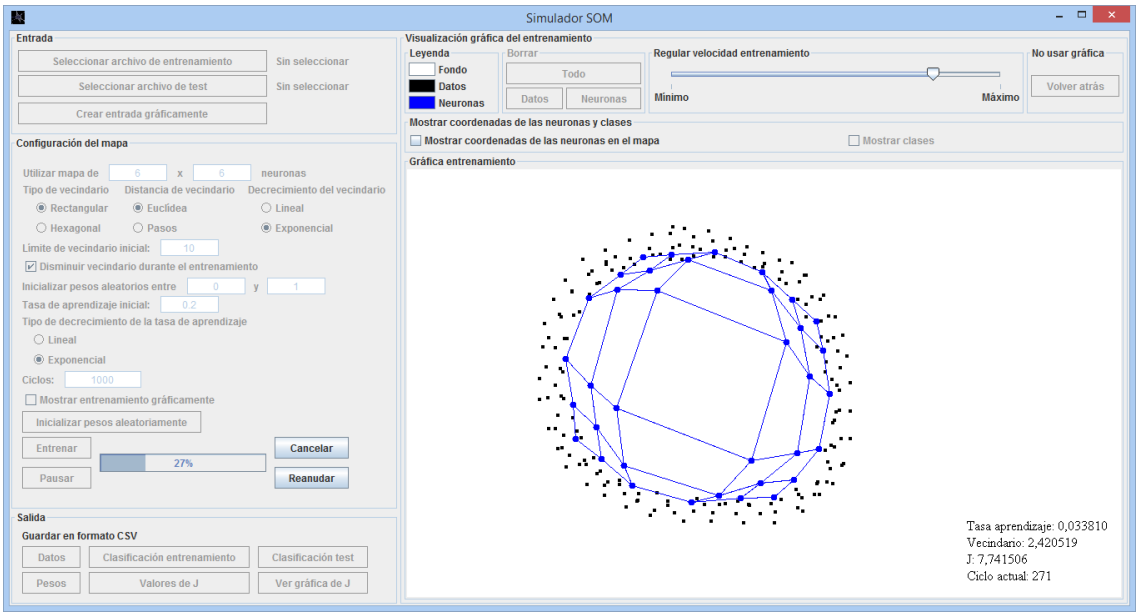


Ilustración 18: ejemplo 2: ciclo 271

Ciclo 271: las neuronas están ajustándose ya a la forma circular, gracias a que el vecindario ha ido disminuyendo permitiendo que se separen.

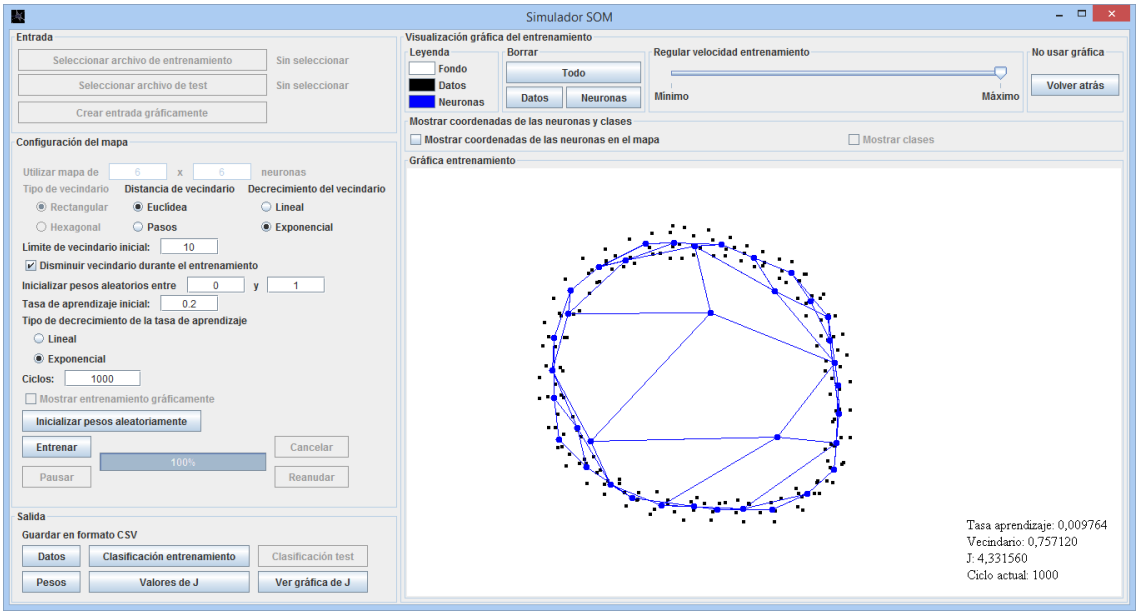


Ilustración 18: ejemplo 2: entrenamiento finalizado

Entrenamiento finalizado: las neuronas están ajustadas a la forma circular. Se puede ver como hay un par de neuronas "sueltas", que no representarían a ningún dato y que han terminado ahí debido a que unas neuronas han "tirado" de ellas hacia un lado y otras hacia otro, haciéndolas terminar en un punto intermedio.

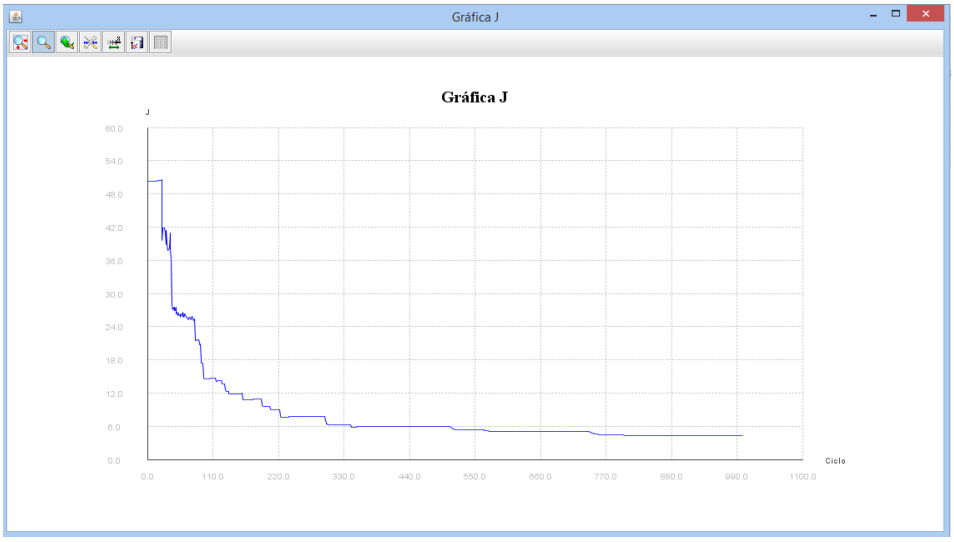


Ilustración 19: ejemplo 2: evolución del error

De nuevo, es posible ver cómo el error ha ido disminuyendo a lo largo del proceso de entrenamiento, encontrándose al principio las bajadas más notables, cuando el mapa sufre los cambios más bruscos.

4.1.3. Ejemplo 3: datos con disposición intencionada

A continuación se exponen otros ejemplos, en los que los datos se han establecido en posiciones que hacen muy sencillo imaginar cómo puede terminar el entrenamiento al utilizarlos:

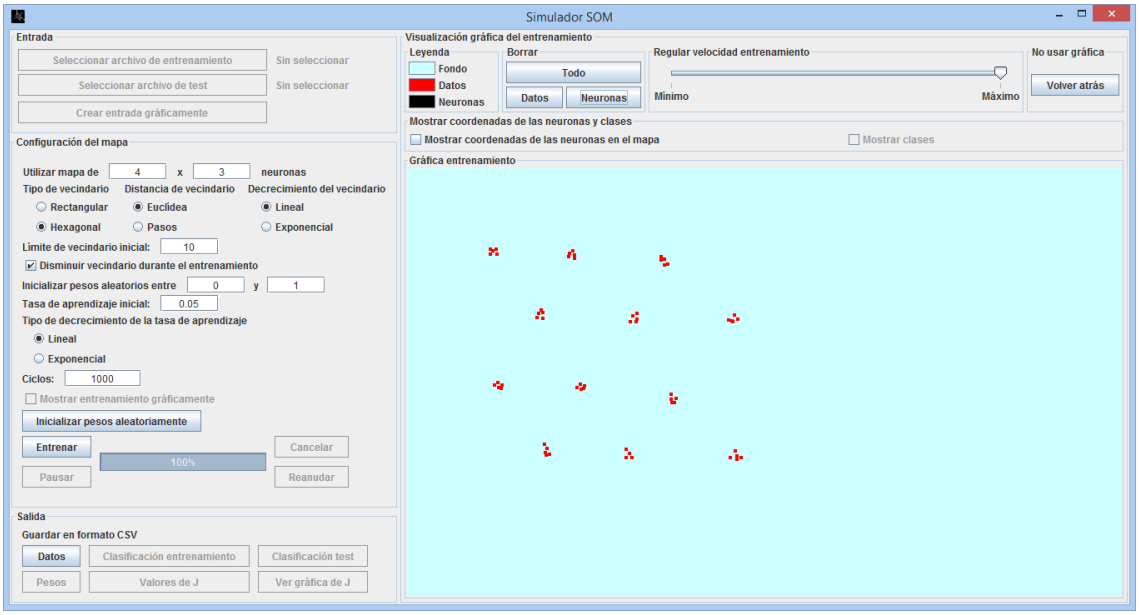


Ilustración 20: ejemplo 3: disposición de los datos

Los datos están formados por pequeños grupos que forman 4 filas y 3 columnas y están dispuestos de tal manera que más o menos se corresponde con una topología hexagonal. Se utilizará un mapa de 4×3 con topología hexagonal. Esta vez se han establecido el decrecimiento del vecindario y la tasa de aprendizaje a lineales. La tasa de aprendizaje tiene un valor de 0.05. El límite de vecindario inicial se ha establecido a 10 (más alto del máximo posible, por lo que se tendrá en cuenta el máximo posible). Se han cambiado los colores de los datos, el fondo y las neuronas.

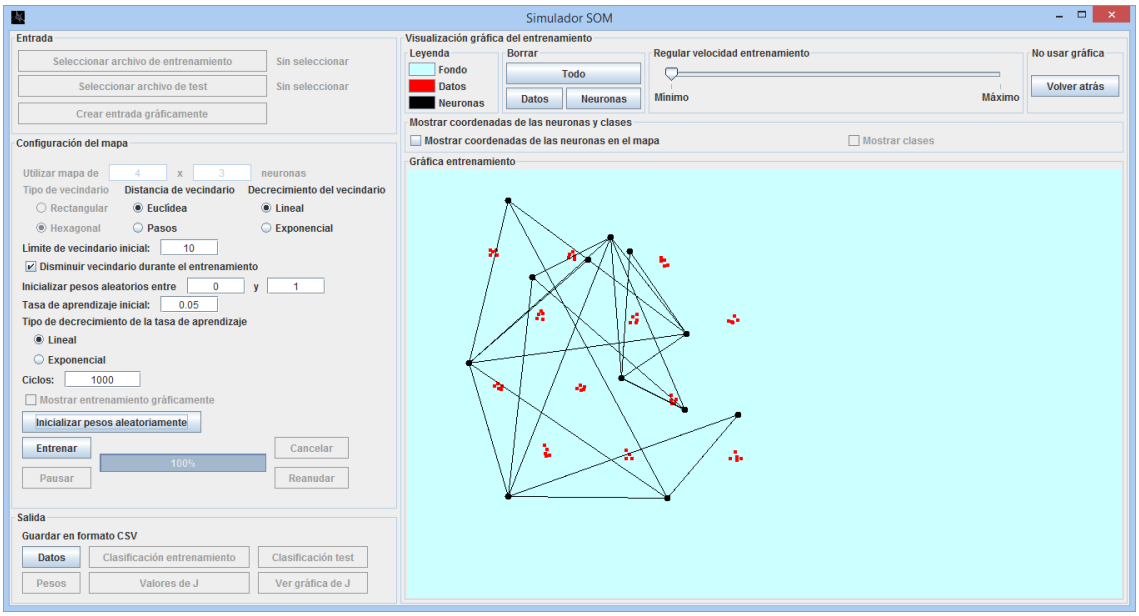


Ilustración 21: ejemplo 3: inicialización aleatoria

Se inicializan aleatoriamente las neuronas.

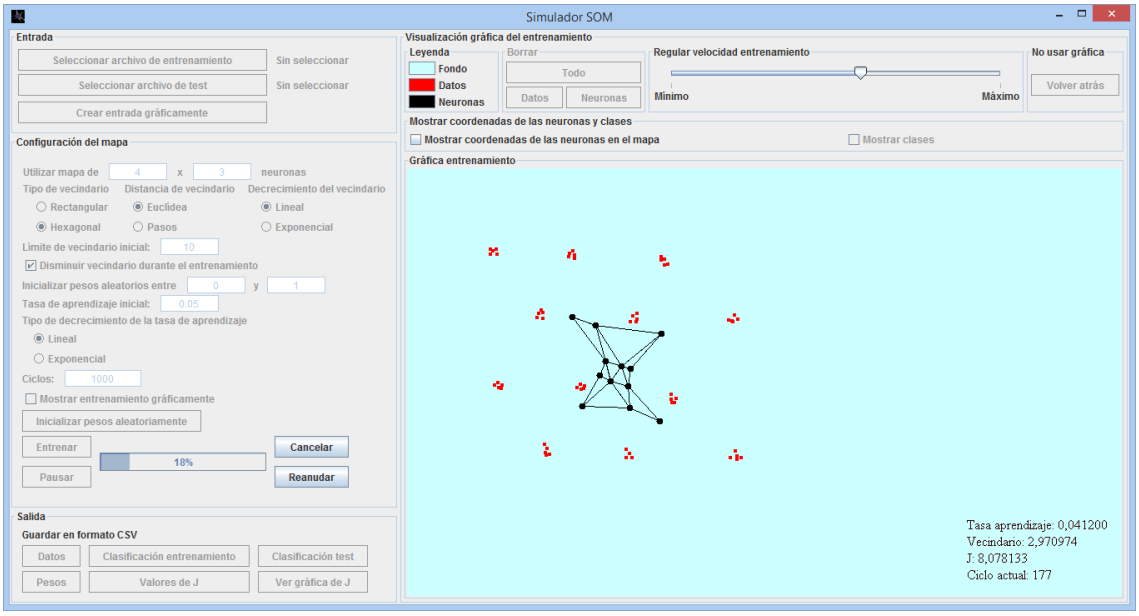


Ilustración 22: ejemplo 3: ciclo 177

Ciclo 177: las neuronas han cambiado mucho respecto a la inicialización aleatoria. Están muy juntas entre sí y cercanas al centro del conjunto de datos.

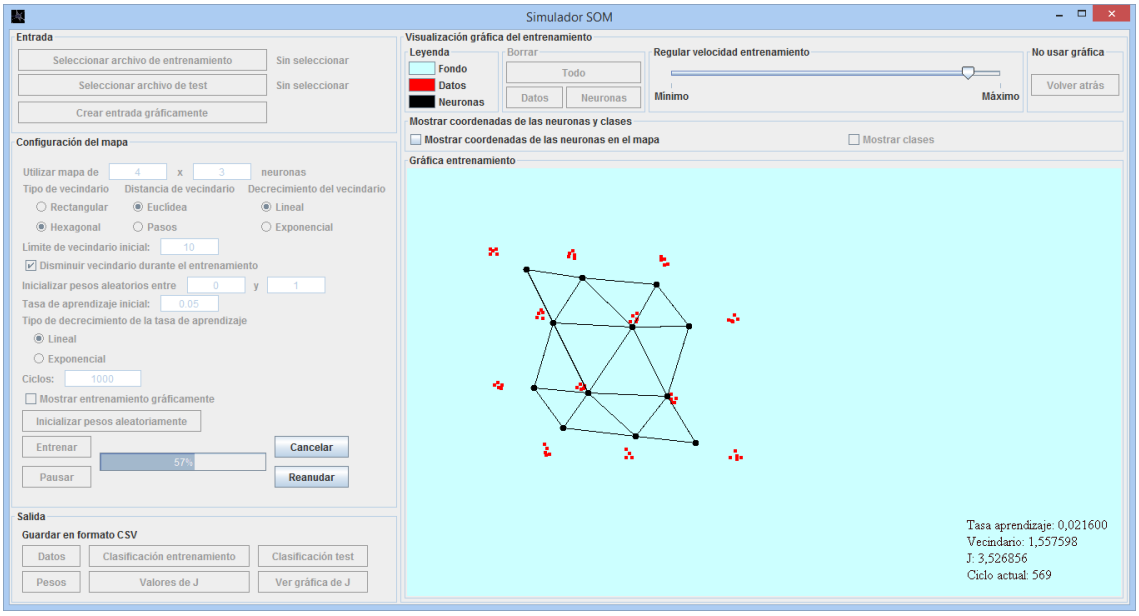


Ilustración 23: ejemplo 3: ciclo 569

Ciclo 569: es posible observar como las neuronas están próximas a ajustarse a los datos.

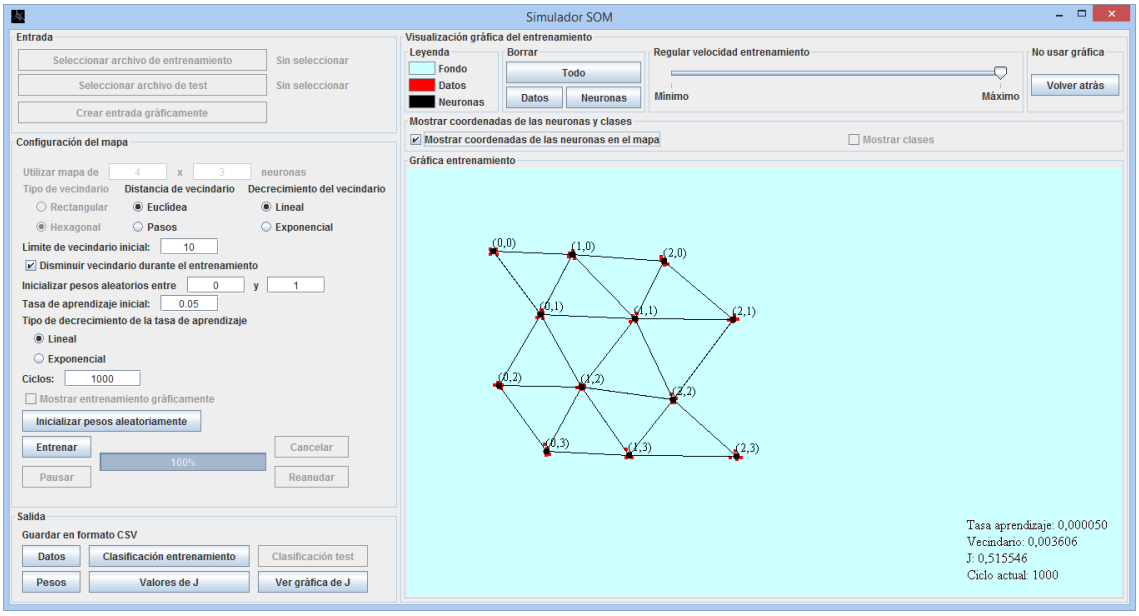


Ilustración 24: ejemplo 3: entrenamiento finalizado

Entrenamiento finalizado: cada neurona se ha posicionado sobre un pequeño grupo de datos, tal como cabía esperar, aunque habría sido posible que la disposición final hubiera sido parecida, pero no tan exacta.

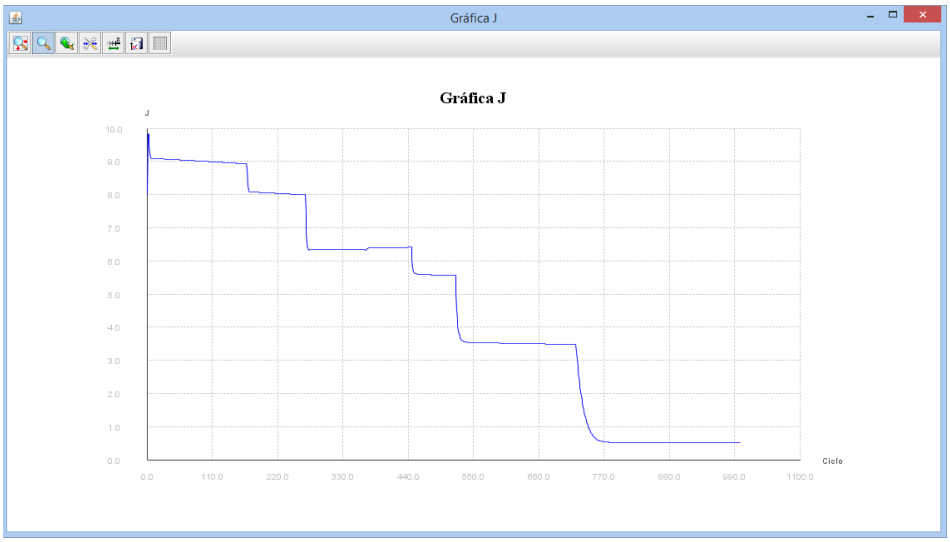


Ilustración 25: ejemplo 3: evolución del error

Evolución del error: se puede observar cómo el error ha disminuido durante el entrenamiento. Puede verse que lo ha hecho de forma bastante escalonada y esta vez no con bajadas mucho mayores al principio que al final, influenciado porque esta vez el descenso de la tasa de aprendizaje y del límite de vecindario eran lineales.

En los ejemplos anteriores, el vecindario inicial era alto, pero ¿qué habría pasado si no se hubiera tenido en cuenta el vecindario y sólo cambiase su posición la neurona ganadora cada vez? En el siguiente ejemplo, con los mismos datos, puede verse.

4.1.4. Ejemplo 4: sin tener en cuenta el vecindario

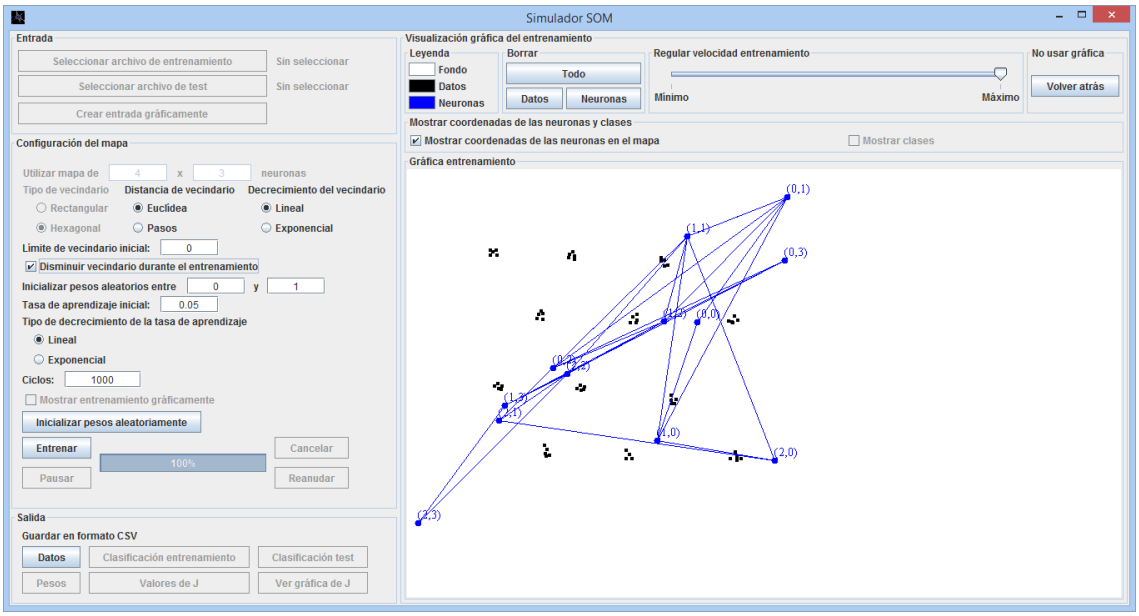


Ilustración 26: ejemplo 4: inicialización aleatoria

Se inicializan aleatoriamente las neuronas y se establece a 0 el límite de vecindario inicial. A continuación, se realiza el entrenamiento:

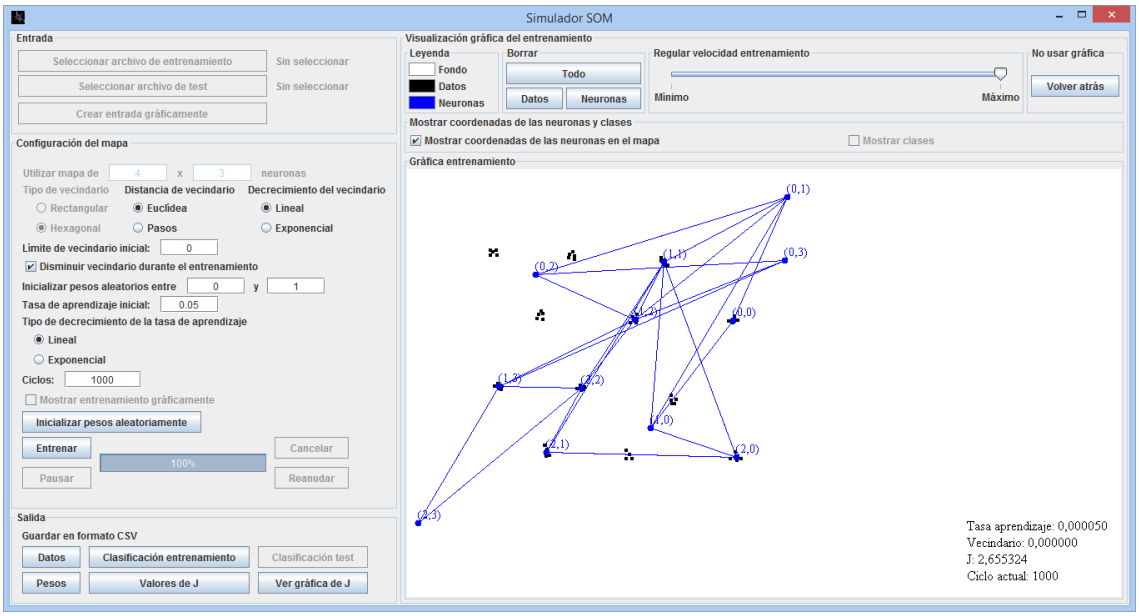


Ilustración 27: ejemplo 4: entrenamiento finalizado

Entrenamiento finalizado: tras 1000 ciclos de entrenamiento, el aspecto tiene mucho parecido con el de la inicialización aleatoria. Además, hay muchas líneas cruzadas entre sí, lo que no ocurre al tener en cuenta el vecindario, ya que, a diferencia de este caso, neuronas cercanas en el mapa tienden a acabar cercanas en el espacio de entrada.

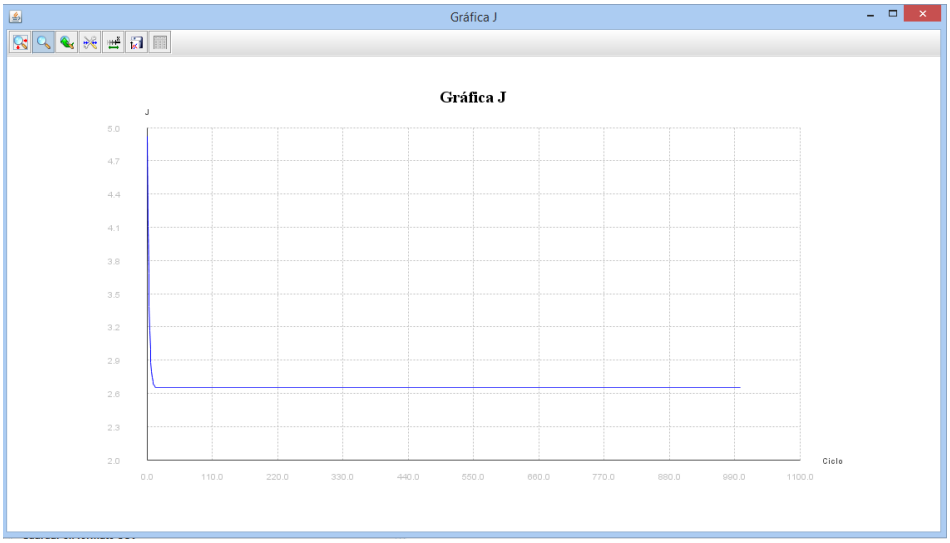


Ilustración 28: ejemplo 4: evolución del error

Evolución del error: el error cometido por la red desciende en los primeros ciclos del entrenamiento y se mantiene prácticamente constante el resto del proceso. No tiene un comportamiento de minimización adecuado.

A continuación veremos lo que ocurre en el caso contrario, en el que el vecindario es muy alto desde el comienzo hasta el final, sin descender a lo largo del entrenamiento.

4.1.5. Ejemplo 5: sin disminuir el vecindario

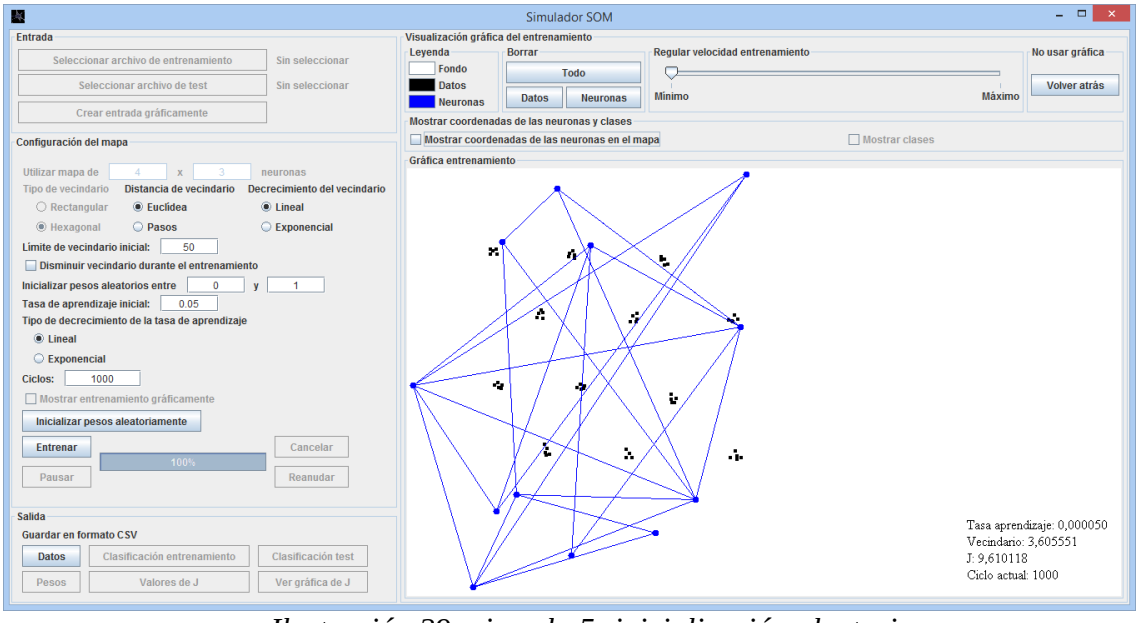


Ilustración 29: ejemplo 5: inicialización aleatoria

Se inicializan aleatoriamente los pesos.

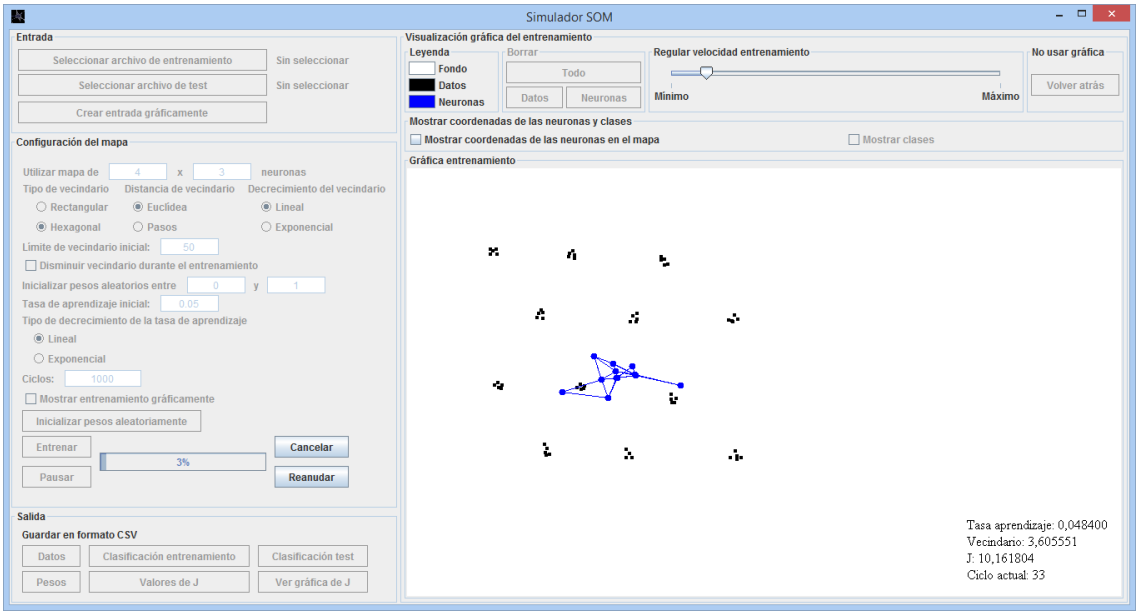


Ilustración 30: ejemplo 5: ciclo 33

Ciclo 33: las neuronas están centradas y muy próximas entre sí.

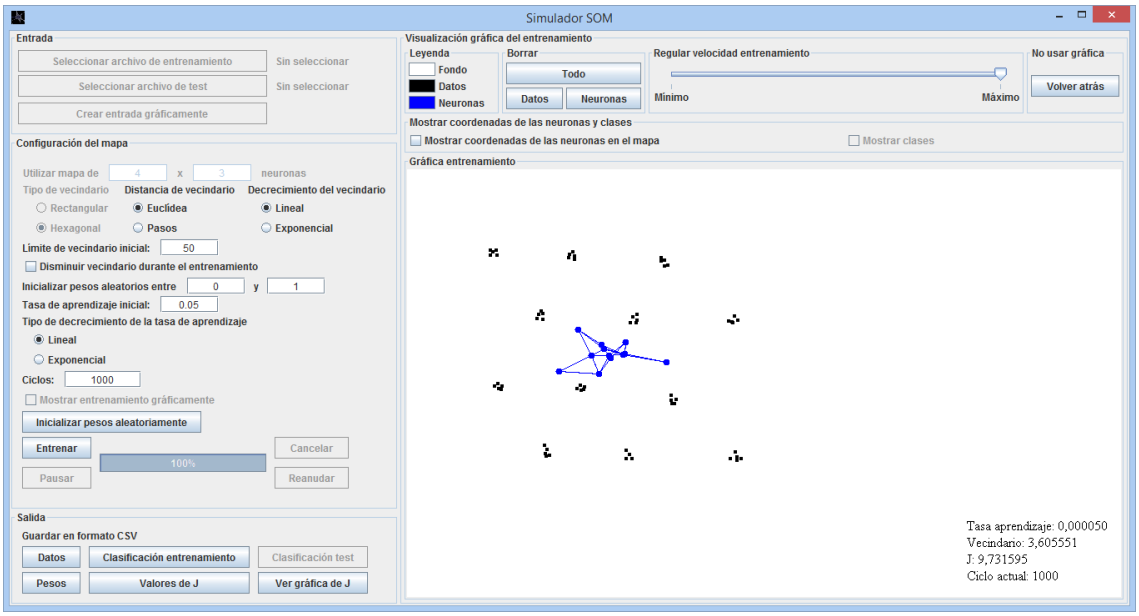


Ilustración 31: ejemplo 5: entrenamiento finalizado

Entrenamiento finalizado: las neuronas continúan estando centradas y muy próximas entre sí. El hecho de que el límite de vecindario sea constantemente alto ha impedido que las neuronas se hayan separado entre sí, ya que cuando una iba a modificar su posición, lo hacía "tirando" de las demás consigo.

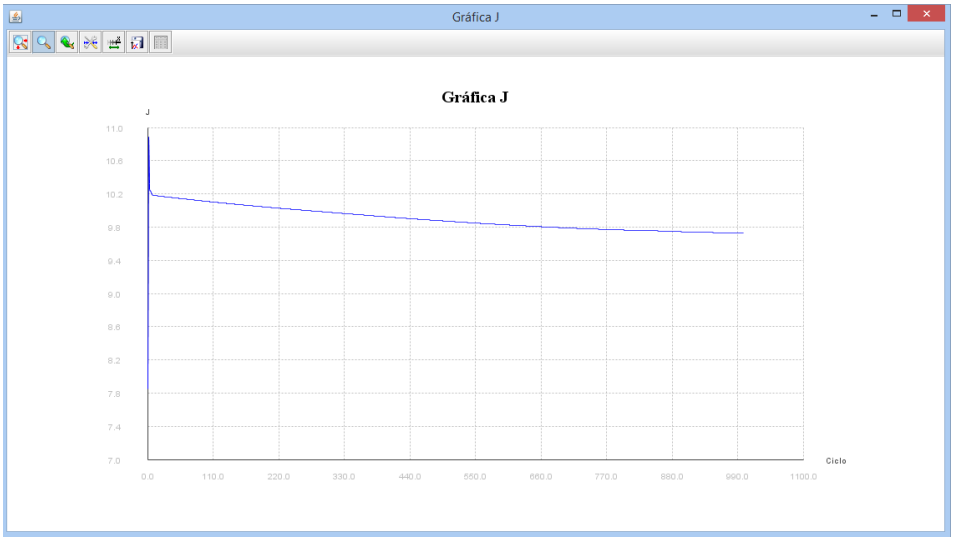


Ilustración 32: ejemplo 5: evolución del error

Evolución del error: el error desciende muy suavemente y muy ligeramente a lo largo del proceso de entrenamiento. De nuevo, no se ha conseguido minimizarlo, como se desearía.

Desde un punto de vista teórico, estos ejemplos sirven para comprobar lo importante que es el concepto de vecindario en los mapas de Kohonen, así como para dar una idea intuitiva de cómo se producen los entrenamientos en función de la disposición de los datos y de los valores de los parámetros.

Desde el punto de vista del proyecto, cabe decir que con estos ejemplos es posible ver cómo se cumplen algunos de los objetivos establecidos inicialmente que motivaron la creación de la aplicación. Se ha visto la sencillez con la que se pueden disponer los datos como se desee sobre la gráfica para utilizarlos como conjunto de entrenamiento. También se ha visto cómo es posible ir viendo e interpretando lo que ocurre durante el entrenamiento, regulando su velocidad o pausándolo si es necesario, tal como se ha hecho para obtener algunas imágenes. Además, tras finalizar el mismo, se puede ver cómo ha evolucionado el error.

4.1.6. Ejemplo 6: usando ficheros y calibración

El siguiente ejemplo, es un resumen del ejemplo que puede encontrarse en el apartado “7.2.6. Ejemplo de uso” del anexo “7.2. Manual de usuario”.

Los datos del fichero de entrenamiento tienen la siguiente distribución:

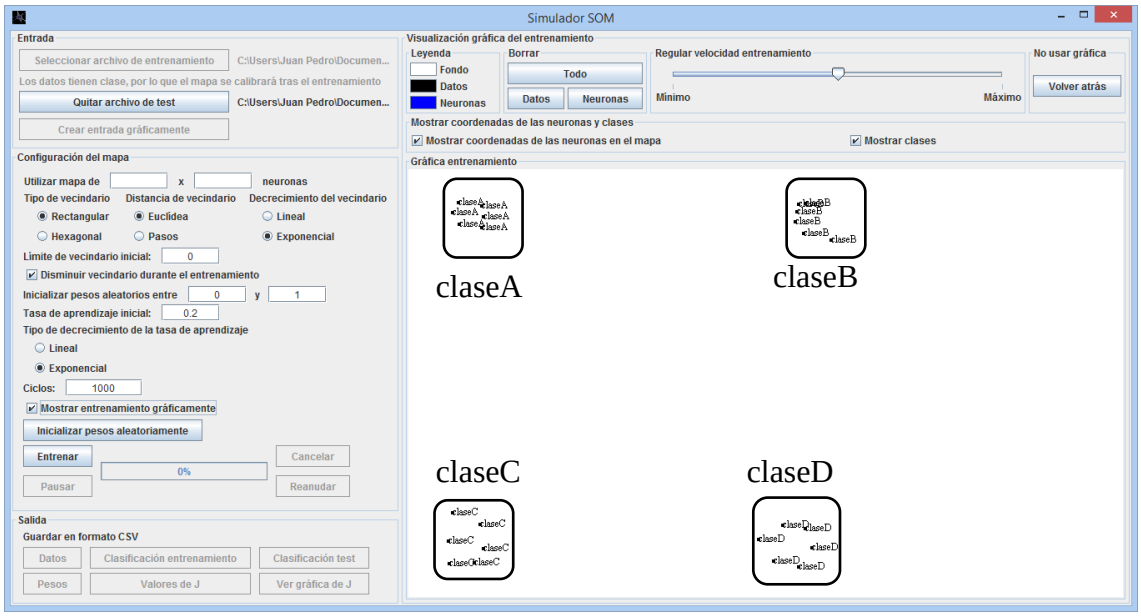


Ilustración 33: ejemplo 6: datos etiquetados

Hay cuatro grupos claramente diferenciados y etiquetados. Al entrenar con un mapa de 2×2, el resultado es el siguiente:

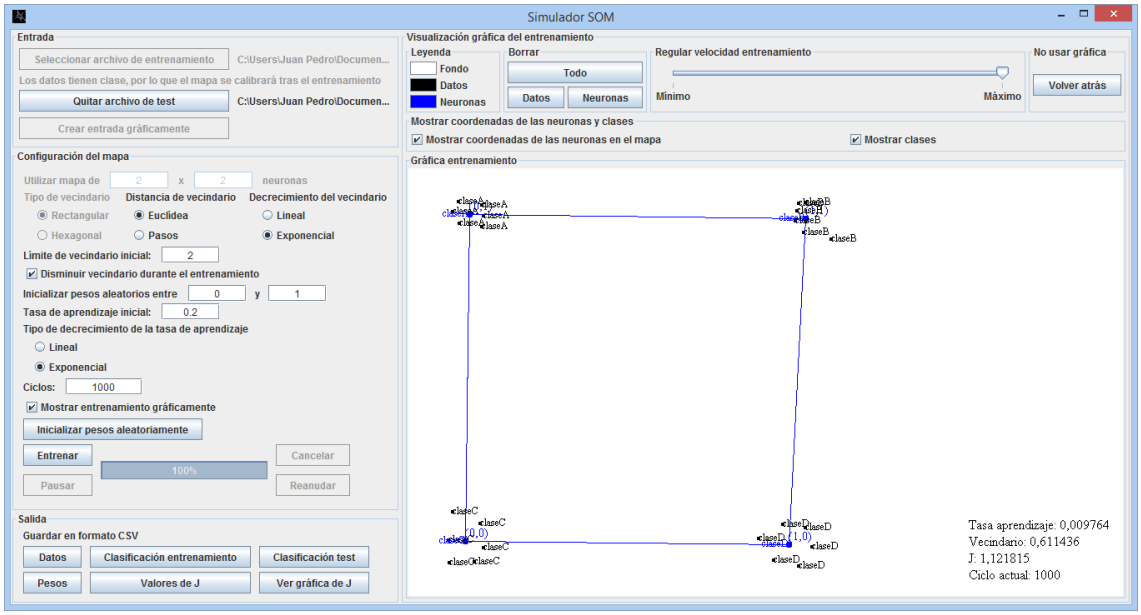


Ilustración 34: ejemplo 6: calibración realizada

Aunque no se aprecia muy bien por superponerse con otras etiquetas, puede verse como al lado de cada neurona, en color azul, se muestra la clase que se le ha asignado tras la calibración (que se realiza automáticamente tras el entrenamiento). Después, dichas clases son usadas en la clasificación que se hace del fichero de entrenamiento y del fichero de test, y son indicadas en el archivo de los pesos.

Este último ejemplo muestra cómo la aplicación también permite el uso de la calibración, con el fin de poder realizar aprendizaje supervisado sin complicar el proceso para el usuario, aunque el entrenamiento sigue siendo no supervisado.

Los ficheros que permite guardar la aplicación, respecto a las clasificaciones de entrenamiento y test son los siguientes:

clasificacionEntrenamiento.csv	clasificacionTest.csv
Línea del fichero,Neurona más cercana,Distancia euclídea entre la neurona y el dato,Clase,¿Bien clasificado? 2,2,0.039460332408966854,claseA,SI 3,2,0.040476610993697565,claseA,SI 4,2,0.032735706239417565,claseA,SI 5,2,0.03551966058791319,claseA,SI 6,2,0.0332352491601035,claseA,SI 7,2,0.04091952560506031,claseA,SI 8,3,0.03752371089301025,claseB,SI 9,3,0.025775012392939278,claseB,SI 10,3,0.03482312914399113,claseB,SI 11,3,0.038522544016403516,claseB,SI 12,3,0.02452108816094987,claseB,SI 13,3,0.07799334519072947,claseB,SI 14,0,0.061638648931200364,claseC,SI 15,0,0.03821317804935971,claseC,SI 16,0,0.07353328799435829,claseC,SI 17,0,0.05439747619217021,claseC,SI 18,0,0.04603940041396799,claseC,SI 19,0,0.05157062225394851,claseC,SI 20,1,0.04714885692807531,claseD,SI 21,1,0.07130153826131333,claseD,SI 22,1,0.05232949156931621,claseD,SI 23,1,0.0560348730684721,claseD,SI 24,1,0.05505981368283453,claseD,SI 25,1,0.053041491044783624,claseD,SI Tasa de aciertos: 100.0%	Línea del fichero,Neurona más cercana,Distancia euclídea entre la neurona y el dato,Clase,¿Bien clasificado? 2,2,0.01168755386150855,claseA,SI 3,3,0.046124087538719175,claseB,SI 4,0,0.01017636818930242,claseC,SI 5,1,0.05824164820673083,claseD,SI Tasa de aciertos: 100.0%

Tabla 57: ejemplo 6: clasificaciones de entrenamiento y test

El fichero de test simplemente contenía un dato de cada clase. El número del campo "Neurona más cercana" se corresponde con el número del campo "Neurona" del archivo de pesos. Cuando se utilizan datos sin clase, los campos "Clase" y "¿Bien clasificado?" no aparecen, como tampoco lo hace la tasa de aciertos. La forma en la que el usuario podría interpretar clases entonces sería como que los datos cuyo campo "Neurona más cercana" es el mismo pertenecen a la misma clase.

pesos.csv
Neurona,Coordenadas neurona (x y),Peso 0,Peso 1,Clase 0,(0 0),0.13316462663497156,0.12707431818690454,claseC 1,(1 0),0.8902411056857554,0.12074861516104983,claseD 2,(0 1),0.14413828623442979,0.8919688585332946,claseA 3,(1 1),0.9289017282384021,0.882476417790367,claseB

Tabla 58: ejemplo 6: pesos

Es posible ver como las neuronas están totalmente identificadas y aparece la clase que se les ha asignado como resultado de la calibración. Si los datos no hubiesen tenido clase, el campo "Clase" no aparecería.

Si hubiese habido neuronas o datos de test a los que no se les hubiese podido asignar una clase, en el campo "Clase" pondría "(Sin clase)".

5. Planificación y presupuesto del trabajo

5.1. Planificación

En la siguiente tabla, se muestra una planificación

Actividad	Fecha de inicio	Duración (días)	Fecha de fin
1. Obtención de requisitos	18/06/2014	3	20/06/2014
2. Selección del lenguaje de programación	21/06/2014	1	21/06/2014
3. Diseño de la aplicación	22/06/2014	5	26/06/2014
4. Implementación	27/06/2014	48	13/08/2014
5. Pruebas	14/08/2014	14	27/08/2014
6. Redacción de la memoria	28/08/2014	20	16/09/2014

Tabla 59: planificación

A continuación, se incluye un diagrama de Gantt representando la planificación:

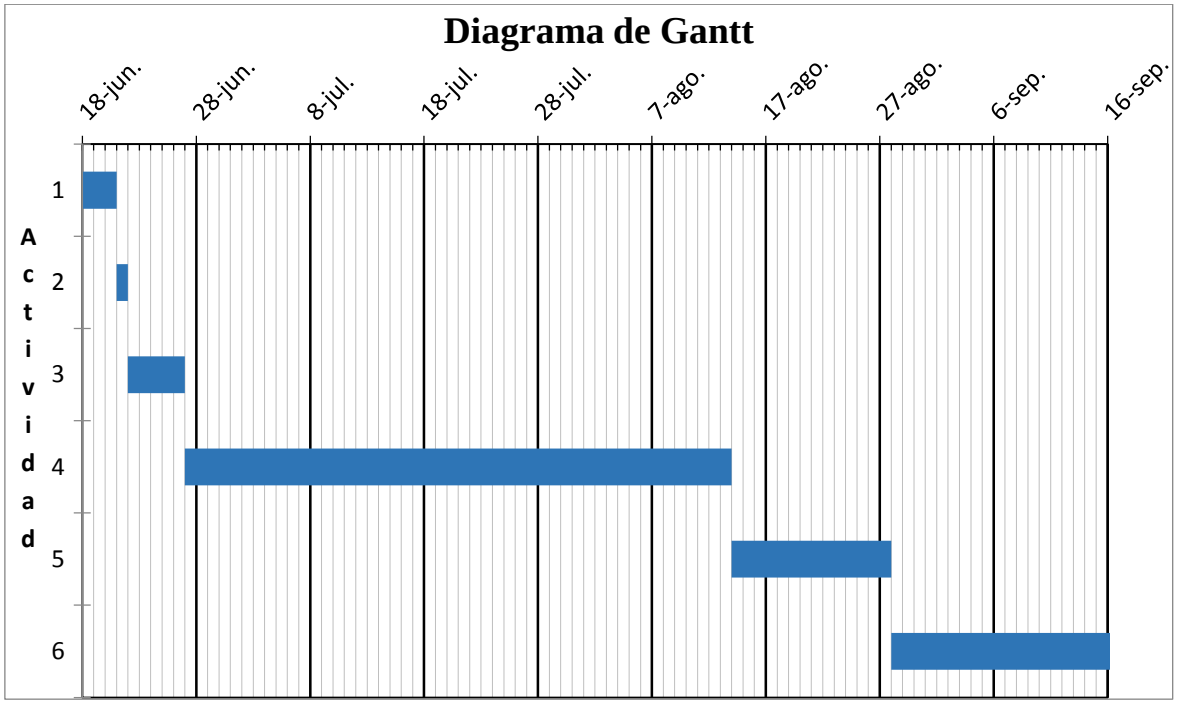


Ilustración 35: diagrama de Gantt, hecho con Microsoft Excel

Aunque se observa una planificación dividida en seis fases, las cuales se realizan de forma secuencial, se prevé que en ciertas ocasiones ocurrirá que hay que volver de una fase a otra anterior, ya que si no se había completado correctamente, conviene modificarla para corregir errores, mejorar la aplicación o conservar la coherencia entre lo realizado en todas las fases.

Como ya se ha comentado anteriormente, no todas las pruebas se hicieron tras terminar toda la implementación, sino que se hacían pruebas a medida que se implementaba, para poder avanzar sabiendo que ciertas cosas están bien implementadas

y que, en caso de errores futuros, fuera más sencillo encontrarlos. Además, si, por ejemplo, durante la implementación se observaba que era necesario modificar algo del diseño para mejorar o corregir lo que se llevaba hecho, esto se hacía, aunque se considerase terminada la fase del diseño.

5.2. Presupuesto

En este apartado, se detalla el presupuesto total del proyecto, incluyendo costes materiales y personales.

Se utilizará la fórmula:

$$\text{Coste imputable} = \frac{\text{Dedicación}}{\text{Período de depreciación}} \cdot \text{Uso dedicado} \cdot \text{Coste sin IVA}$$

En primer lugar, se exponen los costes por hardware. Únicamente se ha utilizado un ordenador portátil:

- Asus A55A-SX465H
  - Procesador: Intel Core i7-3630QM:
    - 2.4 GHz
    - 6 MB de caché
    - 4 núcleos, 8 hilos
  - 8 GB de memoria RAM DDR3.
  - SO: Windows 8.

  Precio sin IVA: 495 €.

Concepto	Dedicación (meses)	Período de depreciación (meses)	Uso dedicado	Coste sin IVA	Coste imputable
Asus A55A-SX465H	3	48	100 %	495 €	30,94 €

Tabla 60: costes imputables por hardware

A continuación, se detallan los costes por software:

Software	Precio de licencia
Windows 8.1	Incluido en el precio del portátil, que venía con Windows 8 y la actualización a Windows 8.1 es gratuita.
Ubuntu 12.10	0 €
Eclipse Juno (versión 4.2, para Windows)	0 €
WhiteStarUML	0 €
Microsoft Office 2013	57,02 € / año

Tabla 61: costes por software

Concepto	Dedicación (meses)	Período de depreciación (meses)	Uso dedicado	Coste sin IVA	Coste imputable
Microsoft Office 2013	3	48	100 %	57,02 €	3,56 €

Tabla 62: costes imputables por software

Para los costes personales, se tendrá en cuenta la planificación anterior, así como que el desarrollador ha sido la misma persona, aunque haya tenido diferentes roles en función de la tarea realizada en cada momento.

Días del proyecto	Horas trabajadas al día (en media)	Coste/hora	Coste total
91	6	18 €	9.828 €

Tabla 63: coste total por personal

Presupuesto total:

Concepto	Coste
Hardware	30,94 €
Software	3,56 €
Personal	9.828 €
Total sin IVA	9.862,50 €
IVA (21%)	2.071,13 €
Total	11.933,63 €

Tabla 64: coste total de la aplicación

El coste total, 11.933,63 € (once mil novecientos treinta y tres con sesenta y tres euros) es el presupuesto necesario para llevar a cabo el proyecto acorde a la planificación, no el que se le daría a un cliente que quisiera comprar la aplicación. En ese caso, sería necesario estudiar la posibilidad de que ocurran contratiempos, así como cuánto beneficio es el que se desea obtener.

6. Conclusiones

6.1. Conclusiones

Este proyecto tenía dos objetivos fundamentales: facilitar el aprendizaje teórico sobre los mapas autoorganizados de Kohonen y servir para entrenarlos con datos reales.

El primer objetivo viene de la necesidad de enseñar a alumnos que no saben lo que son los mapas de Kohonen de una manera lo más intuitiva posible su funcionamiento. De esta forma, será posible hacer que utilicen las ideas adquiridas para entender mejor el significado de las ecuaciones matemáticas que los definen, así como la influencia que tienen los diferentes parámetros.

Este objetivo se considera cumplido, ya que se ha visto cómo es posible añadir datos sobre una gráfica simplemente con clics de ratón, configurar los parámetros y entrenar visualizando qué ocurre. Además, el usuario tiene la opción de guardar los datos que ha generado para cargarlos después en la aplicación.

Se considera un objetivo de gran importancia, ya que aunque existen otras herramientas que permiten visualizar cómo entrena un mapa, generalmente no son todo lo flexibles que podrían respecto a la configuración de los datos de entrada y de los parámetros.

El segundo objetivo consistía en que la aplicación sirviera también para resolver problemas reales, cuyos datos se encuentren en ficheros. Esto implicaba que la aplicación fuera capaz de leer ficheros con un cierto formato. También que fuera capaz de entrenar, pero habiendo cumplido el objetivo anterior, ya lo era. Pero los datos que se añaden gráficamente no tienen clase. Para cumplir satisfactoriamente este objetivo era muy interesante añadir la posibilidad de calibrar el mapa si los datos tenían clase. Ya se ha explicado anteriormente en qué consiste la calibración: asignar una clase (o no) a cada neurona en función de las clases de los datos a los que representa una vez finalizado el entrenamiento, el cual es siempre es no supervisado.

A diferencia del primer objetivo, para el que no existía una aplicación que lo satisficiera tal cual, sí que existen simuladores que permiten trabajar con datos reales, siendo el más representativo “SOM_PAK”.

Este objetivo también se considera satisfecho, ya que la aplicación es capaz de entrenar leyendo de ficheros, calibrar el mapa si los datos tienen clase y utilizar un fichero de test. Además, la aplicación permite guardar un fichero detallado identificando y especificando los pesos de cada neurona del mapa.

A nivel más personal, este proyecto me ha permitido experimentar qué conlleva la realización de un proyecto de software completo, en lugar de prácticas que solamente cubren algunos aspectos. Puesto que la única persona implicada en el mismo además de mí era mi tutor, hacer que diferentes partes del proyecto encajaran era relativamente sencillo, incluso aplicando cambios. Sin embargo, en proyectos de software con equipos de más personas, donde unos realizan una parte mientras otros realizan otra, es fundamental que se coordinen adecuadamente y haya buena comunicación.

6.2. Tendencias futuras

Aunque el trabajo realizado se considera completo y que cumple los objetivos establecidos, siempre es posible aportar mejoras.

Cuando se utiliza algún algoritmo de aprendizaje automático y se dispone de los datos que se quieren utilizar para aprender, lo normal es que éstos no estén de la forma más adecuada para el aprendizaje. A menudo, es necesario realizar un preprocesamiento con ellos. Una de las tareas más habituales del preprocesamiento es la normalización. Este proceso consiste en cambiar los valores de todos los atributos para que estén en el rango [0,1]. Dicho cambio debe hacerse teniendo en cuenta el valor máximo y el valor mínimo que cada atributo puede tomar. De esta forma, aunque haya atributos que por su naturaleza tengan un rango de valores muy amplio, pudiendo ir desde valores pequeños a valores muy altos, mientras que en otros el rango de valores es mucho menor, se consigue que todos sean tenidos en cuenta de forma similar durante el aprendizaje. Sin normalización, estas diferencias entre unos atributos y otros habrían influido en el entrenamiento.

Otra tarea de preprocesamiento de datos muy habitual al trabajar con redes de neuronas artificiales consiste en asignar un valor numérico real a atributos que tienen valor nominal. A una red de neuronas no se le puede presentar como entrada ningún atributo con valor nominal, todos deben ser números reales. Por eso, cuando algún atributo puede tomar valores nominales, éstos son transformados a números reales. Además, como normalmente se utilizan datos normalizados, esta transformación se suele hacer a valores entre 0 y 1, lo más espaciadamente posible (o a veces se codifican usando varios atributos).

Que la aplicación pudiera realizar automáticamente estas tareas sería interesante y podría hacerse de forma que el uso del programa sea prácticamente igual de intuitivo, a pesar de ofrecer más funcionalidad. Existen otras características que también podrían ser interesantes, como la selección de los atributos más relevantes mediante métodos como PCA (Principal Component Analysis) o CFS (Correlation Feature Selection) o la posibilidad de proporcionar alguna información visual de los datos aunque tengan más de dos dimensiones. Sin embargo, éstas se alejan más de los objetivos principales del proyecto.

Otra funcionalidad que podría tenerse en cuenta es permitir ajustar el tamaño con el que se muestran los datos y las neuronas, ya que si la aplicación está en una pantalla o proyector que van a ver varias personas a diferentes distancias, puede que alterar el tamaño haga que lo vean mejor. También podría añadirse la posibilidad de guardar el mapa, con un formato que permita que la propia aplicación pueda cargarlo más tarde.

7. Anexos

En esta sección se incluyen una serie de anexos para detallar más a fondo algunos aspectos de la aplicación.

7.1. Diagrama de clases por partes

Se incluye el diagrama de clases mostrado anteriormente por partes, para una mejor visualización del mismo.

En primer lugar, se incluye el diagrama completo, incluyendo los atributos y las operaciones en las clases:

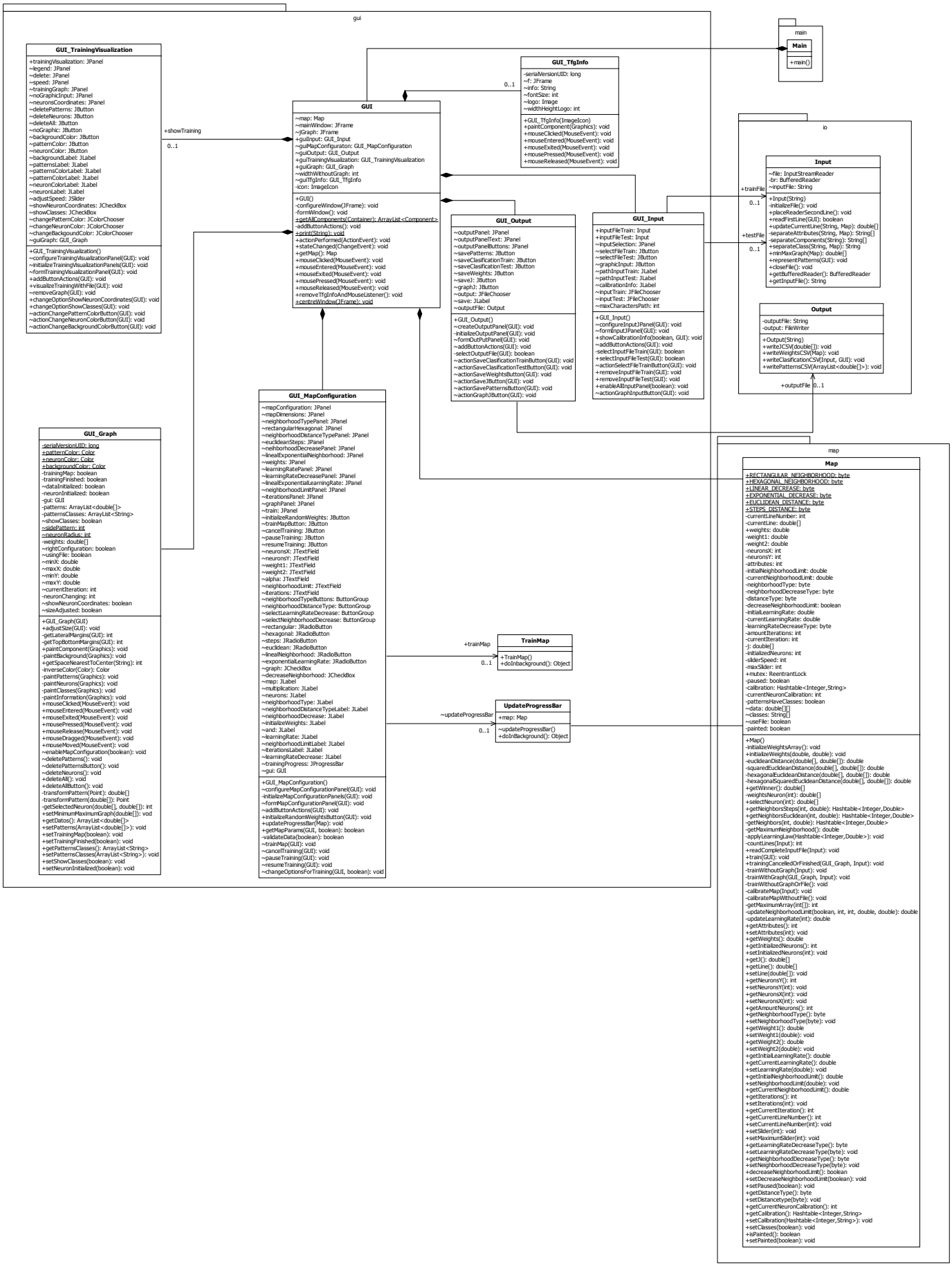
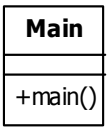


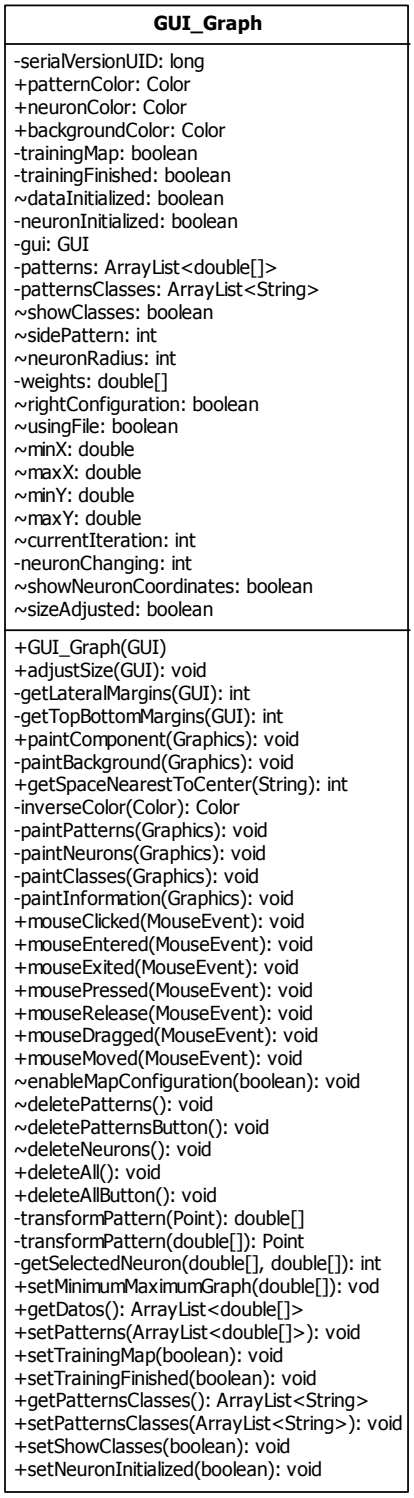
Ilustración 36: diagrama de clases completo en UML

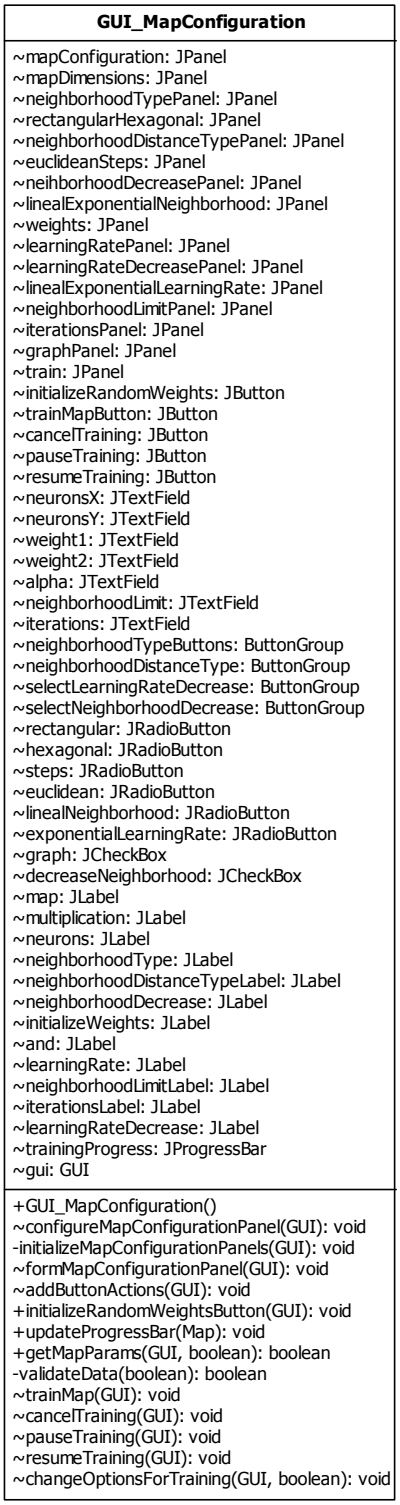
A continuación se muestran todas las clases independientemente, separadas por paquetes, mostrando los atributos y operaciones:

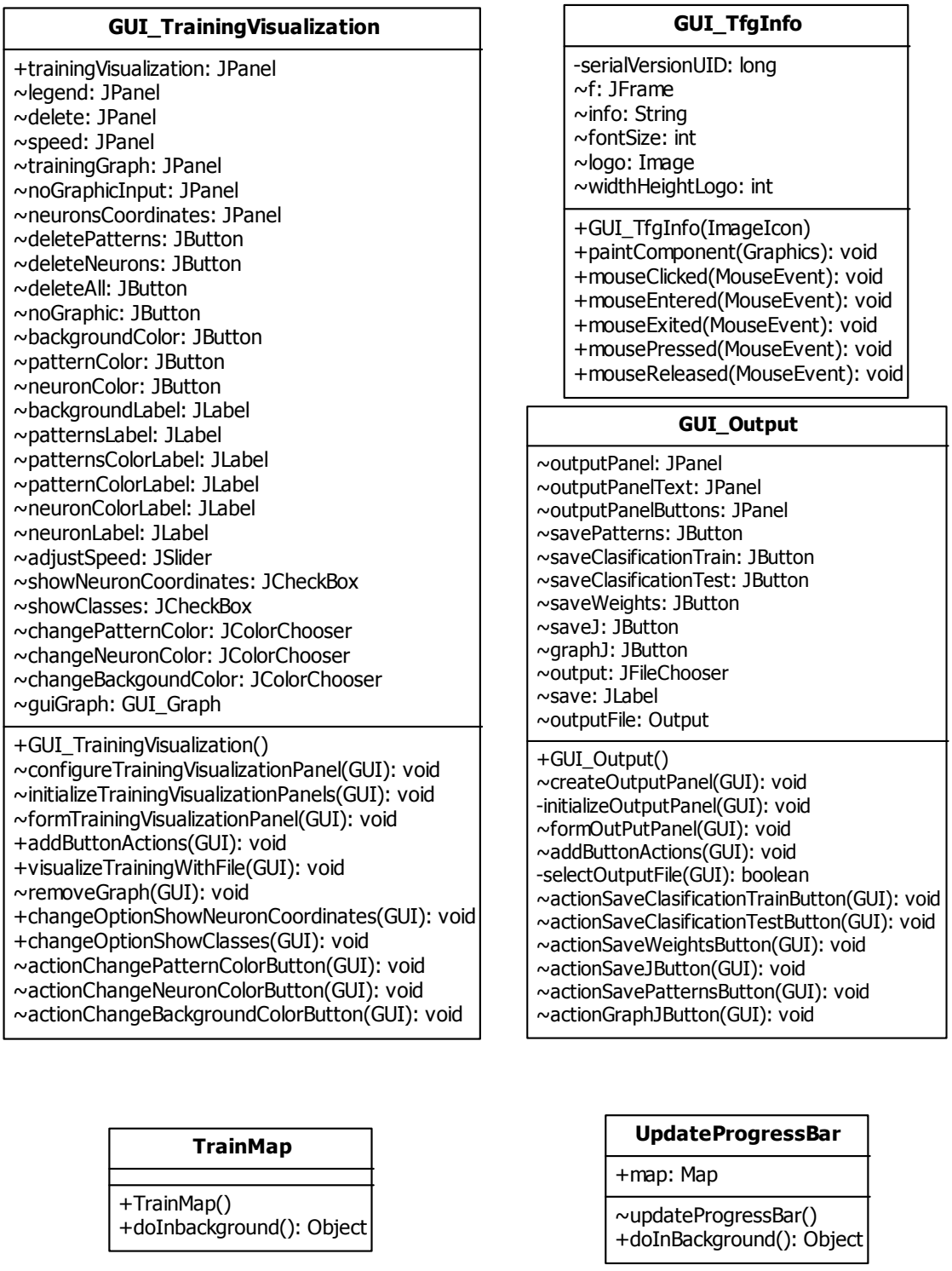
- main

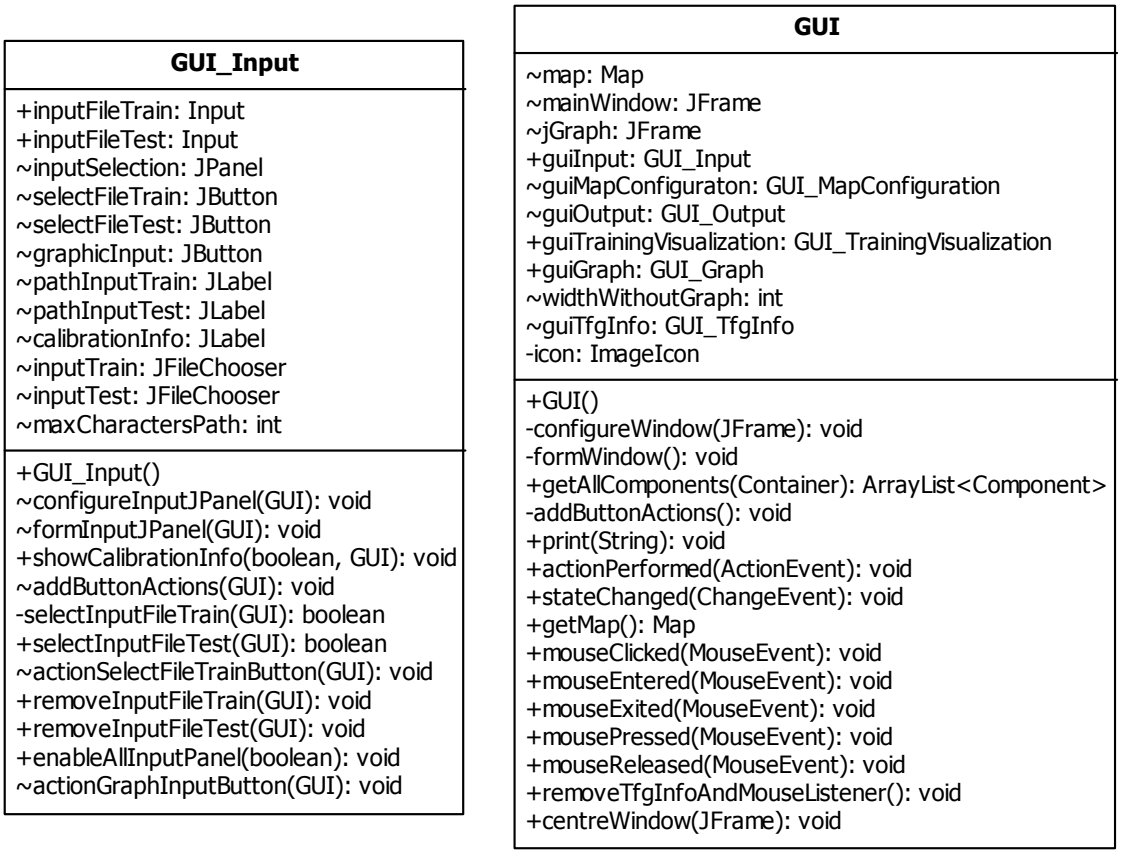


- gui

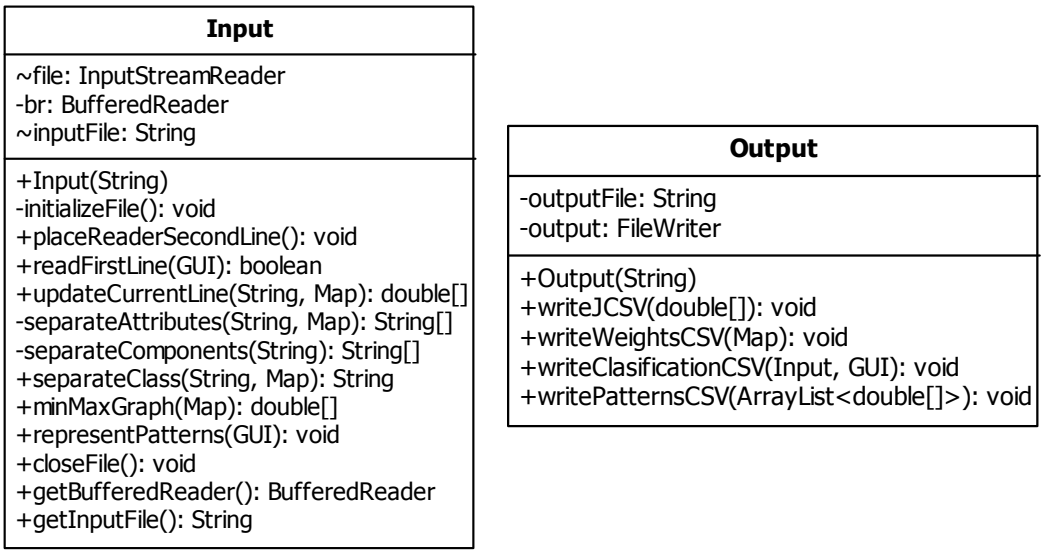





GUI_TrainingVisualization
+trainingVisualization: JPanel
~legend: JPanel
~delete: JPanel
~speed: JPanel
~trainingGraph: JPanel
~noGraphicInput: JPanel
~neuronsCoordinates: JPanel
~deletePatterns: JButton
~deleteNeurons: JButton
~deleteAll: JButton
~noGraphic: JButton
~backgroundColor: JButton
~patternColor: JButton
~neuronColor: JButton
~backgroundLabel: JLabel
~patternsLabel: JLabel
~patternsColorLabel: JLabel
~patternColorLabel: JLabel
~neuronColorLabel: JLabel
~neuronLabel: JLabel
~adjustSpeed: JSlider
~showNeuronCoordinates: JCheckBox
~showClasses: JCheckBox
~changePatternColor: JColorChooser
~changeNeuronColor: JColorChooser
~changeBackgoundColor: JColorChooser
~guiGraph: GUI_Graph
+GUI_TrainingVisualization()
~configureTrainingVisualizationPanel(GUI): void
~initializeTrainingVisualizationPanels(GUI): void
~formTrainingVisualizationPanel(GUI): void
+addButtonActions(GUI): void
+visualizeTrainingWithFile(GUI): void
~removeGraph(GUI): void
+changeOptionShowNeuronCoordinates(GUI): void
+changeOptionShowClasses(GUI): void
~actionChangePatternColorButton(GUI): void
~actionChangeNeuronColorButton(GUI): void
~actionChangeBackgroundColorButton(GUI): void
GUI_TfgInfo
-serialVersionUID: long
~f: JFrame
~info: String
~fontSize: int
~logo: Image
~widthHeightLogo: int
+GUI_TfgInfo(ImageIcon)
+paintComponent(Graphics): void
+mouseClicked(MouseEvent): void
+mouseEntered(MouseEvent): void
+mouseExited(MouseEvent): void
+mousePressed(MouseEvent): void
+mouseReleased(MouseEvent): void
GUI_Output
~outputPanel: JPanel
~outputPanelText: JPanel
~outputPanelButtons: JPanel
~savePatterns: JButton
~saveClasificationTrain: JButton
~saveClasificationTest: JButton
~saveWeights: JButton
~saveJ: JButton
~graphJ: JButton
~output: JFileChooser
~save: JLabel
~outputFile: Output
+GUI_Output()
~createOutputPanel(GUI): void
-initializeOutputPanel(GUI): void
~formOutPutPanel(GUI): void
~addButtonActions(GUI): void
-selectOutputFile(GUI): boolean
~actionSaveClasificationTrainButton(GUI): void
~actionSaveClasificationTestButton(GUI): void
~actionSaveWeightsButton(GUI): void
~actionSaveJButton(GUI): void
~actionSavePatternsButton(GUI): void
~actionGraphJButton(GUI): void
TrainMap
+TrainMap()
+doInbackground(): Object
UpdateProgressBar
+map: Map
~updateProgressBar()
+doInBackground(): Object


GUI_Input
+inputFileTrain: Input
+inputFileTest: Input
~inputSelection: JPanel
~selectFileTrain: JButton
~selectFileTest: JButton
~graphicInput: JButton
~pathInputTrain: JLabel
~pathInputTest: JLabel
~calibrationInfo: JLabel
~inputTrain: JFileChooser
~inputTest: JFileChooser
~maxCharactersPath: int
+GUI_Input()
~configureInputJPanel(GUI): void
~formInputJPanel(GUI): void
+showCalibrationInfo(boolean, GUI): void
~addButtonActions(GUI): void
-selectInputFileTrain(GUI): boolean
+selectInputFileTest(GUI): boolean
~actionSelectFileTrainButton(GUI): void
+removeInputFileTrain(GUI): void
+removeInputFileTest(GUI): void
+enableAllInputPanel(boolean): void
~actionGraphInputButton(GUI): void
GUI
~map: Map
~mainWindow: JFrame
~jGraph: JFrame
+guiInput: GUI_Input
~guiMapConfiguraton: GUI_MapConfiguration
~guiOutput: GUI_Output
+guiTrainingVisualization: GUI_TrainingVisualization
+guiGraph: GUI_Graph
~widthWithoutGraph: int
~guiTfgInfo: GUI_TfgInfo
-icon: ImageIcon
+GUI()
-configureWindow(JFrame): void
-formWindow(): void
+getAllComponents(Container): ArrayList<Component>
-addButtonActions(): void
+print(String): void
+actionPerformed(ActionEvent): void
+stateChanged(ChangeEvent): void
+getMap(): Map
+mouseClicked(MouseEvent): void
+mouseEntered(MouseEvent): void
+mouseExited(MouseEvent): void
+mousePressed(MouseEvent): void
+mouseReleased(MouseEvent): void
+removeTfgInfoAndMouseListener(): void
+centreWindow(JFrame): void

- io


Input
~file: InputStreamReader
-br: BufferedReader
~inputFile: String
+Input(String)
-initializeFile(): void
+placeReaderSecondLine(): void
+readFirstLine(GUI): boolean
+updateCurrentLine(String, Map): double[]
-separateAttributes(String, Map): String[]
-separateComponents(String): String[]
+separateClass(String, Map): String
+minMaxGraph(Map): double[]
+representPatterns(GUI): void
+closeFile(): void
+getBufferedReader(): BufferedReader
+getInputFile(): String
Output
-outputFile: String
-output: FileWriter
+Output(String)
+writeJCSV(double[]): void
+writeWeightsCSV(Map): void
+writeClasificationCSV(Input, GUI): void
+writePatternsCSV(ArrayList<double[]>): void

- map

Clase "Map"	
Atributos	Operaciones
+RECTANGULAR_NEIGHBORHOOD: byte +HEXAGONAL_NEIGHBORHOOD: byte +LINEAR_DECREASE: byte +EXPONENTIAL_DECREASE: byte +EUCLIDEAN_DISTANCE: byte +STEPS_DISTANCE: byte -currentLineNumber: int -currentLine: double[] +weights: double -weight1: double -weight2: double -neuronsX: int -neuronsY: int -attributes: int -initialNeighborhoodLimit: double -currentNeighborhoodLimit: double -neighborhoodType: byte -neighborhoodDecreaseType: byte -distanceType: byte -decreaseNeighborhoodLimit: boolean -initialLearningRate: double -currentLearningRate: double -learningRateDecreaseType: byte -amountIterations: int -currentIteration: int -j: double[] -initializedNeurons: int -sliderSpeed: int -maxSlider: int +mutex: ReentrantLock -paused: boolean -calibration: Hashtable<Integer,String> -currentNeuronCalibration: int -patternsHaveClasses: boolean ~data: double[][] ~classes: String[] ~useFile: boolean -painted: boolean	+Map() -initializeWeightsArray(): void +initializeWeights(double, double): void -euclideanDistance(double[], double[]): double -squaredEuclideanDistance(double[], double[]): double -hexagonalEuclideanDistance(double[], double[]): double -hexagonalSquaredEuclideanDistance(double[], double[]): double +getWinner(): double[] -weightsNeuron(int): double[] +selectNeuron(int): double[] +getNeighborsSteps(int, double): Hashtable<Integer,Double> +getNeighborsEuclidean(int, double): Hashtable<Integer,Double> -getNeighbors(int, double): Hashtable<Integer,Double> -getMaximumNeighborhood(): double -applyLearningLaw(Hashtable<Integer,Double>): void -countLines(Input): int +readCompleteInputFile(Input): void +train(GUI): void +trainingCancelledOrFinished(GUI_Graph, Input): void -trainWithoutGraph(Input): void -trainWithGraph(GUI_Graph, Input): void -trainWithoutGraphOrFile(): void -calibrateMap(Input): void -calibrateMapWithoutFile(): void -getMaximumArray(int[]): int -updateNeighborhoodLimit(boolean, int, int, double, double): double -updateLearningRate(int): double +getAttributes(): int +setAttributes(int): void +getWeights(): double +getInitializedNeurons(): int +setInitializedNeurons(int): void +getJ(): double[] +getLine(): double[] +setLine(double[]): void +getNeuronsY(): int +setNeuronsY(int): void +getNeuronsX(int): void +setNeuronsX(int): void +getAmountNeurons(): int +getNeighborhoodType(): byte +setNeighborhoodType(byte): void +getWeight1(): double +setWeight1(double): void +getWeight2(): double +setWeight2(double): void +getInitialLearningRate(): double +getCurrentLearningRate(): double +setLearningRate(double): void +getInitialNeighborhoodLimit(): double +setNeighborhoodLimit(double): void +getCurrentNeighborhoodLimit(): double +getIterations(): int +setIterations(int): void +getCurrentIteration(): int +getCurrentLineNumber(): int +setCurrentLineNumber(int): void +setSlider(int): void +setMaximumSlider(int): void +getLearningRateDecreaseType(): byte +setLearningRateDecreaseType(byte): void +getNeighborhoodDecreaseType(): byte +setNeighborhoodDecreaseType(byte): void +decreaseNeighborhoodLimit(): boolean +setDecreaseNeighborhoodLimit(boolean): void +setPaused(boolean): void +getDistanceType(): byte +setDistancetype(byte): void +getCurrentNeuronCalibration(): int +getCalibration(): Hashtable<Integer,String> +setCalibration(Hashtable<Integer,String>): void +setClasses(boolean): void +isPainted(): boolean +setPainted(boolean): void

Tabla 65: atributos y operaciones de la clase "Map"

7.2. Manual de usuario

El siguiente anexo que se adjunta es un manual de usuario de la aplicación:

Simulador SOM

Manual de usuario

Índice

7.2.1. Introducción

La aplicación "Simulador SOM" ha sido desarrollada como un Trabajo de Fin de Grado de la Universidad Carlos III de Madrid, realizado por el alumno Juan Pedro García Ruiz y dirigido por el profesor José María Valls Ferrán.

Se trata de un simulador de mapas autoorganizados de Kohonen, un tipo de red de neuronas artificiales que realiza aprendizaje no supervisado. Su principal objetivo es servir como herramienta didáctica mostrando gráficamente cómo se realiza el proceso de aprendizaje, lo cual ayuda a entender el funcionamiento general de este tipo de redes y la influencia de sus parámetros. Además, sirve también para realizar entrenamientos leyendo ficheros, tanto de entrenamiento como de test, y escribiendo los resultados en otros ficheros.

Para poder ejecutar correctamente la aplicación, es necesario disponer de *Java Runtime Environment (JRE)*.

7.2.2. Presentación de la aplicación

Nada más iniciar la aplicación, se puede ver lo siguiente:

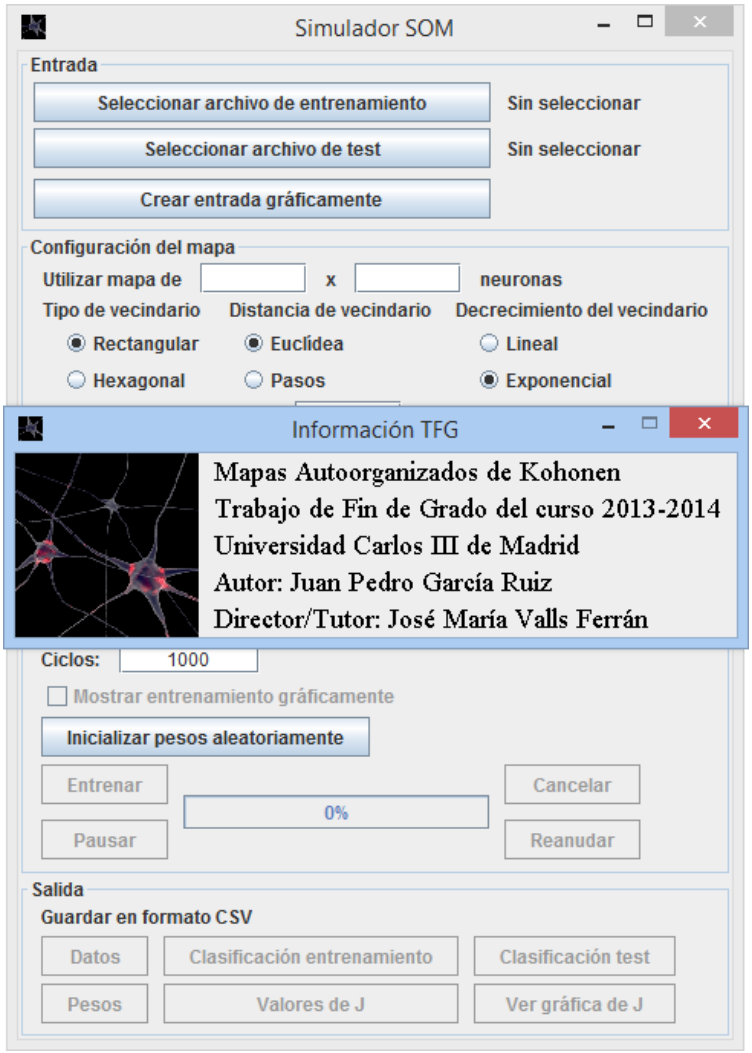


Ilustración 37: aplicación recién iniciada

La ventana que aparece en primer plano en la ilustración es simplemente informativa y puede quitarse cerrándola o haciendo clic sobre ella o sobre cualquier parte de la aplicación. Una vez cerrada, se ve la ventana principal de la aplicación:

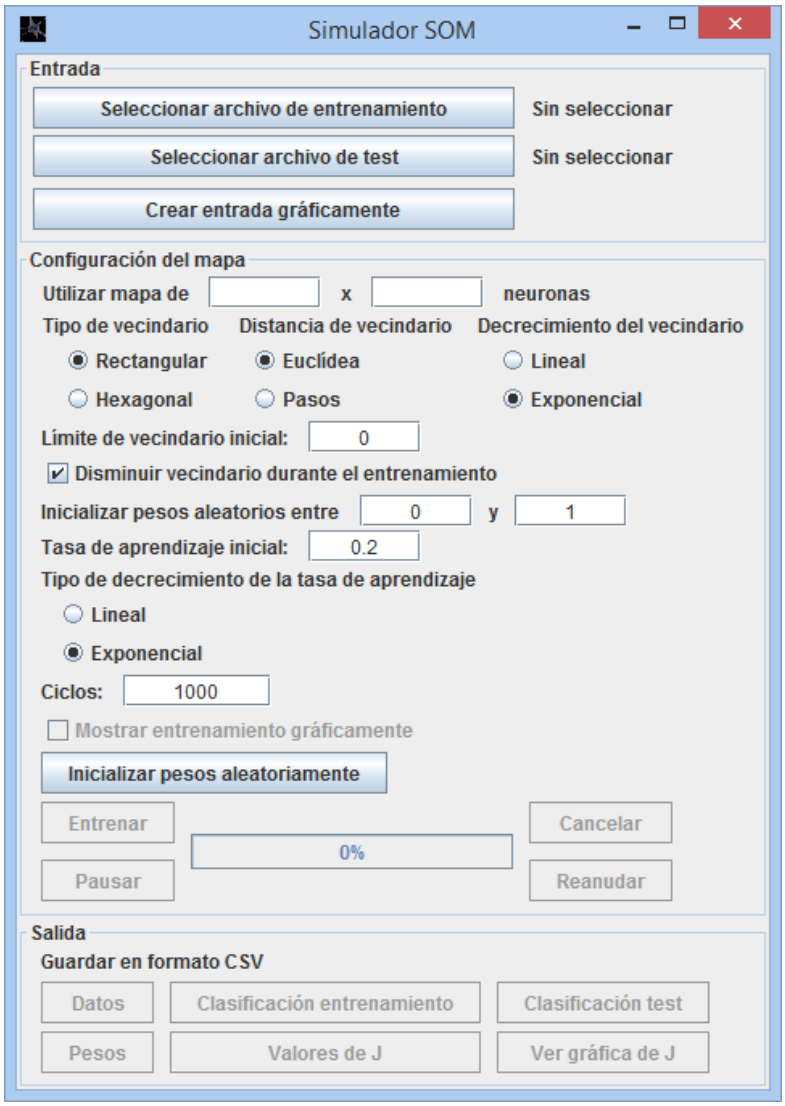


Ilustración 38: ventana principal de la aplicación

Se puede ver que la aplicación está dividida en tres partes: entrada, configuración del mapa y salida.

7.2.3. Entrada

La parte de la entrada dispone de tres botones:

- Seleccionar archivo de entrenamiento. Abre un explorador para seleccionar el archivo de entrenamiento que puede tener cualquier extensión. A su derecha hay un texto que especifica el *path* del archivo de entrenamiento seleccionado. Si no hay ninguno seleccionado pone "Sin seleccionar". Si el *path* no se muestra entero, es posible dejar la flecha del ratón encima para verlo. Si se selecciona un archivo con datos que tienen clase, bajo el botón aparecerá el texto: "Los datos tienen clase, por lo que el mapa se calibrará tras el entrenamiento".

- Seleccionar archivo de test. Al igual que para el archivo de entrenamiento, abre un explorador para seleccionar un archivo de test que puede tener cualquier extensión. A su derecha aparece el *path* de dicho archivo.
- Crear entrada gráficamente. Al pulsarlo, la interfaz cambia añadiendo una parte para visualizar el entrenamiento. Impide usar archivos de entrenamiento y test, pero permite generar un conjunto de datos de entrenamiento mediante clics de ratón.

7.2.3.1. Formato de los ficheros de entrada

Aunque pueden tener cualquier extensión, los ficheros de entrenamiento y test deben tener un formato específico para poder ser leídos correctamente por la aplicación. El formato es el siguiente:

- Una primera línea que contiene únicamente un entero que indica la cantidad de atributos que tienen los datos.
- Tantas líneas como datos haya con el siguiente formato:
  - <Atributo1><sep><Atributo2><sep>…<sep><AtributoN>[<Clase>]
- Los atributos son números reales cuyo separador decimal es el punto (.).
- El separador puede ser:
  - Una coma.
  - Una sucesión de espacios en blanco (espacios o tabulaciones) precedidos, o no, por una coma.
- La clase puede aparecer o no. En caso de aparecer puede tener valor numérico o nominal. Es importante que no contenga caracteres que puedan interpretarse como un separador, es decir, ni coma, ni espacios, ni tabulaciones.

Ejemplos de ficheros de entrada válidos:

Ejemplos sin clase	Ejemplos con clase
3 0.287,0.442,0.62 0.7,0.125,0.54786 0.145,0.555,0.211 0.34,0,0.651	2 1.12e-8 0.005 claseA 3.25474 4.254 claseB 15.245498 23 claseC
5 0.256, 4.215, 2, 0.254, 0 0.1, 2, 12, 1.2, 6.457	4 1 2 3 4 0 5 6 7 8 1

Tabla 66: ejemplos de formato de posibles ficheros

7.2.3.2. Entrada gráfica

Al pulsar sobre el botón “Crear entrada gráficamente”, la interfaz cambia y pasa a mostrarse de este modo:

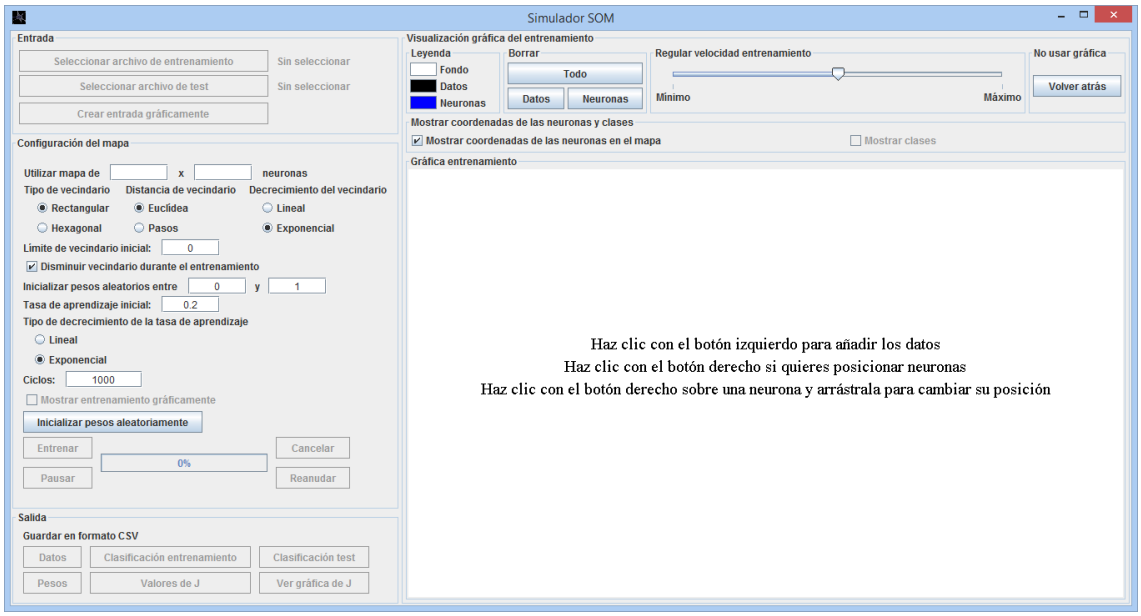


Ilustración 39: ventana principal con la parte gráfica añadida

Se han deshabilitado los botones que permitían seleccionar archivos de entrenamiento y test. En caso de que hubiese habido archivos seleccionados antes de pulsar el botón, se habrían ignorado.

En la parte añadida se pueden ver distintas secciones:

- Leyenda. Contiene botones que especifican cuál es y permiten cambiar el color del fondo, los datos y las neuronas.
- Borrar. Permite borrar los datos añadidos, las neuronas o todo.
- Regular velocidad entrenamiento. Permite hacer que el entrenamiento transcurra más despacio o más rápido.
- No usar gráfica. Contiene un botón que permite quitar todo lo relacionado con la gráfica y seleccionar archivos de nuevo.
- Mostrar coordenadas de las neuronas y clases. Permite mostrar o no las coordenadas de las neuronas en el mapa (no en el espacio de entrada). La opción de mostrar clases está inhabilitada, ya que los datos que se crearán no tendrán clase asignada.
- Gráfica entrenamiento. Sobre ella se puede:
  - Añadir datos haciendo clic con el botón izquierdo del ratón.
  - Posicionar neuronas haciendo clic con el botón izquierdo del ratón. Para ello, es necesario haber definido correctamente las dimensiones del mapa en el panel de "Configuración del mapa".
  - Cambiar la posición de las neuronas arrastrándolas con el botón derecho.

  La esquina inferior izquierda se considera el punto (0,0) y la superior izquierda el (0,1). El resto de la gráfica tiene en cuenta esa proporción.

7.2.4. Configuración del mapa

En el panel de configuración del mapa se encuentran los parámetros de la red que se deben configurar previamente a un entrenamiento:

- Dimensiones. Es necesario especificar, con números naturales, cuántas filas y cuantas columnas tendrá el mapa.
- Tipo de vecindario. Puede ser rectangular o hexagonal.
  Ni el tipo de vecindario ni las dimensiones pueden ser modificados si se está mostrando una red en la gráfica.
- Distancia de vecindario. Indica cómo debe medirse la distancia entre neuronas:
  - Euclídea. Mide la distancia euclídea de las neuronas en el mapa (no en el espacio de entrada).
  - Pasos. Cuenta el número mínimo de enlaces entre dos neuronas.
- Decrecimiento del vecindario. Indica si durante el entrenamiento el vecindario debe decrecer lineal o exponencialmente. Este decrecimiento sólo se produce si está activada la opción "Disminuir vecindario durante el entrenamiento".
- Límite de vecindario inicial. Número real que indica, inicialmente, hasta qué distancia pueden encontrarse las neuronas vecinas de la ganadora que también actualizarán su posición. Establecerlo a cero (o un número inferior a uno) implica no tener en cuenta el vecindario. Poner un número superior a la distancia entre las dos neuronas más lejanas entre sí es equivalente a poner dicha distancia.
- Disminuir vecindario durante el entrenamiento. Aunque no es lo más habitual, es posible hacer que el vecindario no disminuya durante el entrenamiento quitando el *tick* de esta opción. Se trata de una opción añadida con el fin de ayudar a comprender cómo influye el vecindario.
- Dos números reales entre los que inicializar los pesos aleatoriamente.
- Si el decrecimiento de la tasa de aprendizaje debe ser lineal o exponencial.
- Tasa de aprendizaje. Número real que indica cuánto vale la tasa de aprendizaje al inicio. Debe estar comprendida entre 0 y 1.
- Ciclos. Entero positivo que indica cuántos ciclos será entrenada la red.

Los parámetros que sean números reales llevarán el punto (.) como separador decimal.

También contiene algunas opciones más:

- Mostrar entrenamiento gráficamente. Es una opción que se habilita cuando se selecciona un archivo de entrenamiento con dos atributos. Si se activa, muestra los datos en la gráfica. Es posible ver sobre la gráfica a qué clase pertenece cada dato pulsando sobre el botón "Mostrar clases" en la sección de "Mostrar coordenadas de las neuronas y clases" de "Visualización gráfica del entrenamiento". La escala de la gráfica dependerá del conjunto de datos, ya que se hace que se muestren todos.
- Inicializar pesos aleatoriamente. Puede usarse, de manera opcional, para inicializar los pesos de la red aleatoriamente entre los valores especificados anteriormente.
- Entrenar. Permite entrenar la red. Si los pesos no han sido inicializados manualmente, se inicializarán al comienzo del entrenamiento.
- Pausar. Permite pausar el entrenamiento.
- Reanudar. Permite reanudar un entrenamiento pausado.
- Cancelar. Permite cancelar el entrenamiento.

- Barra de progreso. Permite al usuario saber qué porcentaje del entrenamiento ha sido realizado ya.

7.2.5. Salida

Es posible guardar en un fichero en formato CSV:

- Datos. El usuario puede guardar los datos de la gráfica si estos no se corresponden con lo leído en un fichero de entrenamiento. Se almacena un fichero que puede ser leído por la aplicación, con la siguiente estructura:
  - La primera línea contiene únicamente el número 2, ya que al estar en la gráfica los datos tienen dos dimensiones.
  - Tantas líneas como datos hay en la gráfica con las coordenadas X e Y de los datos separadas por una coma.

  Ejemplo de fichero:

```
2
0.25,0.1
0.589,0.147
0.48,0.9
0.7,1
```

Tabla 67: ejemplo de fichero almacenado por la aplicación

- Clasificación entrenamiento. Se almacena un fichero con la clasificación de los datos de entrenamiento al final del mismo, con la siguiente estructura:
  - Una primera línea indicando qué significan los campos separados por comas de las siguientes líneas.
  - Tantas líneas como datos de entrenamiento hubiese, con los siguientes campos separados por comas:
    - Línea del fichero. Indica a qué dato hace referencia la línea. La forma de señalarlo es con un número que indica en qué línea del fichero de entrenamiento (en caso de no haberlo usado, del fichero que se guarda al pulsar sobre el botón "Datos" se encuentra el dato).
    - Neurona más cercana. Indica en qué neurona se clasifica el dato. La neurona está representada con un entero entre 0 y N-1, siendo N el total de neuronas. La forma de saber las coordenadas de la neurona en el mapa y sus pesos (así como la clase si los datos la tenían) es comprobarlo en el fichero que se almacena con el botón "Pesos".
    - Distancia euclídea entre la neurona y el dato.

    Si los datos tienen clase, existen otros dos campos:
    - Clase. Se indica qué clase se le asigna al dato.
    - ¿Bien clasificado?. Indica si la clase asignada al dato es la correcta o no.
  - Por último, si los datos tienen clase, aparece una línea que indica la tasa de aciertos.
- Clasificación test. El fichero que se guarda sigue la misma estructura que el de clasificación de entrenamiento.
- Pesos. Permite almacenar un fichero con la siguiente estructura:
  - Una línea indicando qué significan los campos de las siguientes líneas.

    - Tantas líneas como neuronas tenga el mapa entrenado, con los siguientes campos, separados por comas:
        - Neurona. Número que indica qué neurona es. Se trata de un entero entre 0 y N-1, siendo N el total de neuronas.
        - Coordenadas de la neurona en el mapa.
        - Pesos de la neurona. Tendrá tantos pesos como atributos los datos.
        - Clase. Este campo sólo aparece si los datos de entrenamiento tienen clase. En este campo, las neuronas que no representan a ningún patrón y, por tanto, no pueden tener clase, aparecen con "(Sin clase)". Es importante que el fichero de datos no tenga ninguna clase cuyo nombre es "(Sin clase)", para evitar errores.
- Valores de J. Almacena un fichero que especifica cuál ha sido el sumatorio de distancias euclídeas entre cada dato y la neurona que lo representa para cada ciclo, dato al que se ha llamado "J", por abreviar. Es posible ver una gráfica mostrando esta información con el botón "Ver gráfica de J". Esta visualización se hace gracias al uso de la librería "JMathPlot", disponible en https://sites.google.com/site/mulabsltd/products/jmathplot, con la siguiente licencia:

Tabla 68: licencia de "JMathPlot"

7.2.6. Ejemplo de uso

A continuación, se expone un ejemplo sencillo del uso de la aplicación:

1. Se utilizan los siguientes ficheros:

train.csv	test.csv
2 0.105,0.897,claseA 0.117,0.922,claseA 0.119,0.871,claseA 0.172,0.914,claseA 0.177,0.887,claseA 0.172,0.862,claseA 0.929,0.92,claseB 0.91,0.9,claseB 0.924,0.848,claseB 0.914,0.918,claseB 0.905,0.877,claseB 0.99,0.834,claseB 0.096,0.0779,claseC 0.095,0.129,claseC 0.105,0.195,claseC 0.169,0.168,claseC 0.177,0.113,claseC 0.154,0.0799,claseC 0.877,0.166,claseD 0.82,0.133,claseD 0.854,0.083,claseD 0.914,0.07,claseD 0.945,0.115,claseD 0.928,0.158,claseD	2 0.148,0.903,claseA 0.883,0.887,claseB 0.125,0.121,claseC 0.832,0.121,claseD

Tabla 69: ficheros utilizados en el ejemplo

2. En la sección de "Configuración del mapa" se activa la opción "Mostrar entrenamiento gráficamente:

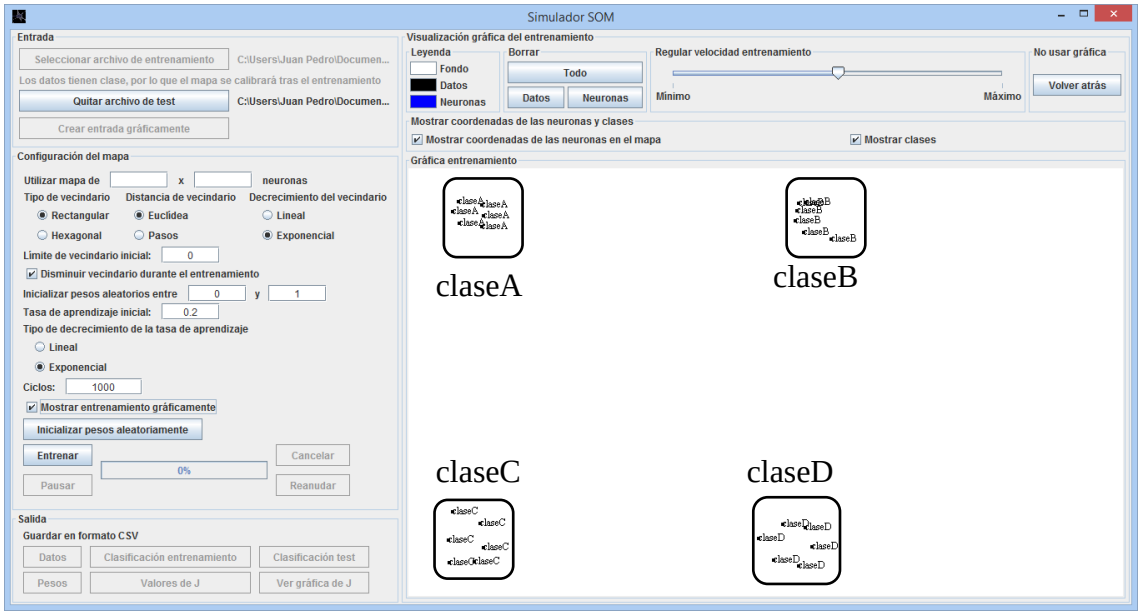


Ilustración 40: gráfica con datos para entrenar

Se puede observar que cada dato de una clase está muy cerca de los datos de su misma clase y lejos de los de otras clases.

3. Se define el mapa de 2×2 y de límite de vecindario se establece 2. El resto de parámetros se conservan como son por defecto.

4. Se entrena el mapa, obteniendo lo siguiente:

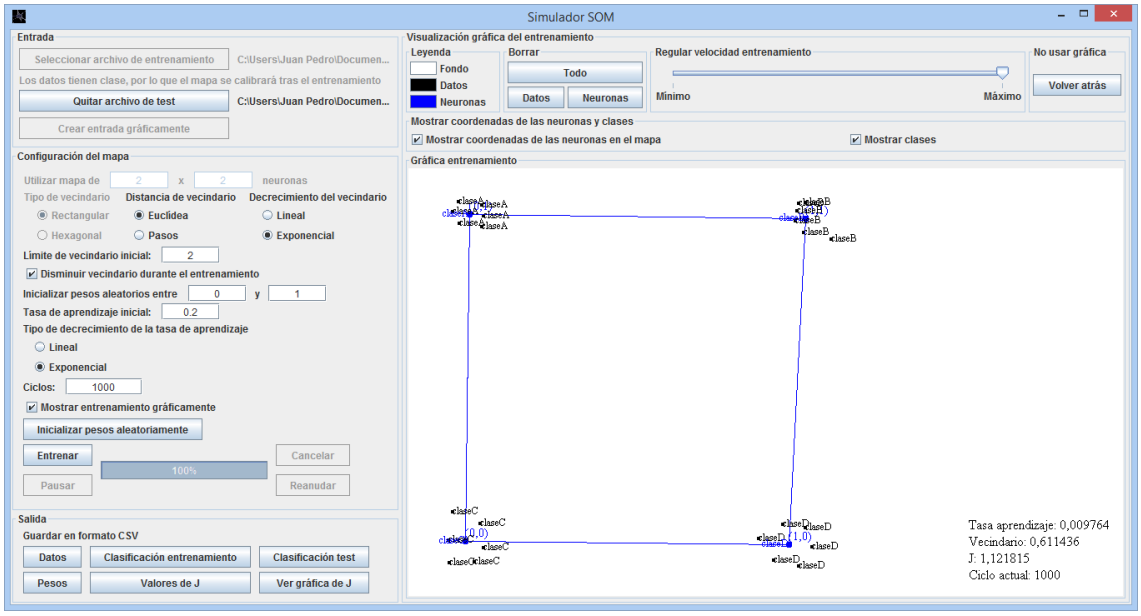


Ilustración 41: gráfica con datos y entrenamiento finalizado

Se puede observar que al terminar el entrenamiento se les ha asignado una clase a las neuronas.

5. Se almacenan los ficheros de clasificación de entrenamiento, de test y el fichero de pesos. Su contenido, en este caso, es el siguiente (no tiene por qué coincidir exactamente aunque hayas seguido los pasos):

clasificacionEntrenamiento.csv	clasificacionTest.csv
Línea del fichero,Neurona más cercana,Distancia euclídea entre la neurona y el dato,Clase,¿Bien clasificado? 2,2,0.039460332408966854,claseA,SI 3,2,0.040476610993697565,claseA,SI 4,2,0.032735706239417565,claseA,SI 5,2,0.03551966058791319,claseA,SI 6,2,0.0332352491601035,claseA,SI 7,2,0.04091952560506031,claseA,SI 8,3,0.03752371089301025,claseB,SI 9,3,0.025775012392939278,claseB,SI 10,3,0.03482312914399113,claseB,SI 11,3,0.038522544016403516,claseB,SI 12,3,0.02452108816094987,claseB,SI 13,3,0.07799334519072947,claseB,SI 14,0,0.061638648931200364,claseC,SI 15,0,0.03821317804935971,claseC,SI 16,0,0.07353328799435829,claseC,SI 17,0,0.05439747619217021,claseC,SI 18,0,0.04603940041396799,claseC,SI 19,0,0.05157062225394851,claseC,SI 20,1,0.04714885692807531,claseD,SI 21,1,0.07130153826131333,claseD,SI 22,1,0.0523294915693162 1,claseD,SI 23,1,0.0560348730684721,claseD,SI 24,1,0.05505981368283453,claseD,SI 25,1,0.053041491044783624,claseD,SI Tasa de aciertos: 100.0%	Línea del fichero,Neurona más cercana,Distancia euclídea entre la neurona y el dato,Clase,¿Bien clasificado? 2,2,0.01168755386150855,claseA,SI 3,3,0.046124087538719175,claseB,SI 4,0,0.01017636818930242,claseC,SI 5,1,0.05824164820673083,claseD,SI Tasa de aciertos: 100.0%

Tabla 70: resultados del ejemplo almacenados por la aplicación

pesos.csv
Neurona,Coordenadas neurona (x y),Peso 0,Peso 1,Clase 0,(0 0),0.13316462663497156,0.12707431818690454,claseC 1,(1 0),0.8902411056857554,0.12074861516104983,claseD 2,(0 1),0.14413828623442979,0.8919688585332946,claseA 3,(1 1),0.9289017282384021,0.882476417790367,claseB

Tabla 71: pesos almacenados por la aplicación

En los archivos "clasificacionEntrenamiento.csv" y "clasificacionTest.csv", el campo "Neurona más cercana" se corresponde con el campo "Neurona" del archivo "pesos.csv".

Si se altera de alguna forma algún dato y/o neurona se deja de mostrar la gráfica, los datos relacionados con el entrenamiento realizado se pierden y ya no es posible almacenar estos ficheros.

7.3. Short description of the work in English

7.3.1. Introduction

7.3.1.1. Motivation

Neural networks are mathematical models constructed from examples and they allow to solve a large number of real problems. In Computer Engineering teachings, there are subjects such as Artificial Intelligence or Artificial Neural Networks in which pupils have to fully understand how these neural network models work. Although they are described perfectly by using mathematical equations, they also require a more intuitive understanding. In general, teachers are faced with the difficulty of transmitting an intuitive view of these models to pupils, i.e., pupils imagine and visualize what happens when a neural network is being trained.

It is important to understand the meaning of the mathematical equations that describe the models, but it is also essential to understand intuitively how these models work, and that is much easier if it is possible to see what happens while models are being constructed, i.e., while neural networks are being trained. Of course, this view will be able to be done only in some simplified situations, when patterns can be represented with two dimensions and they are not excessively numerous, but it allows pupils to understand intuitively their basis and it will not be a problem to imagine the same model extended to spaces with more dimensions.

Holding conversations with Artificial Neural Networks teachers, they told me that it is necessary a simple and intuitive application that allows to see different parts of neural networks learning, and concretely of a special and very important kind of network: Kohonen self-organizing maps, very used to solve problems that require unsupervised learning. I have thought mathematical equations were very complicated, because of unknowing what means every component and why they were related in that way, but when I understood the intuitive idea and how the map works, understanding and applying equations is much easier. Moreover, when they are applied making mistakes, sometimes it is possible to know it checking the results and thinking in Kohonen maps behavior.

It is not easy to find software that allows to see this process, even though there are some applets that make possible to see some things, but they are very basic and they do not take into account some of the most relevant parameters.

This is the fundamental motivation to develop this work. Furthermore, the application developed also has to serve to work with real patterns with more dimensions, even though they cannot to be seen.

7.3.1.2. Objectives

7.3.1.2.1. General objectives

Then, some general objectives of the application are mentioned, related to the ideas that motivated it:

- The application has to serve like a didactic tool to see what happens when Kohonen self-organizing maps is being trained.
- The application also will be able to be used with real patterns with an arbitrary quantity of dimensions, although they could not be viewed.

7.3.1.2.2. Partial objectives

Then, some partial objectives that the application must achieve are exposed, related to application implementation:

- The application must be intuitive. User will be able to place patterns in the plane in a simple way using the mouse.
- The application has to be robust: if an error happens it has to inform the user without stop working.
- Even though learning algorithm is unsupervised, sometimes real patterns are labeled. In that case, it is possible to take advantage of it to label new patterns.
- The application will work with different operative systems (Windows and Linux).
- When model had been constructed using training data, it will be able to be validated using test data.
- An indicative error evolution will be shown related to training.

7.3.2. Problem approach

7.3.2.1. State of the art

7.3.2.1.1. Kohonen maps description

A Kohonen map or SOM (Self-Organizing Map) is a kind of artificial neural network for unsupervised learning. Its name is related to its author, Teuvo Kohonen (born 11 July 1934), Emeritus Professor of the Academy of Finland. He has published over 300 research papers and four monography books. His fifth book is on digital computers. [5]

Kohonen maps can be used to solve different problems, such as:

- clustering [2]
- data mining
- process analysis, diagnostics, monitoring, and control
- biomedical applications, including diagnostic methods and data analysis in bioinformatics
- data analysis in commerce, industry, macroeconomics, and finance [3]

This type of artificial neural networks has two layers: the input layer (F1) and the competition layer (F2).

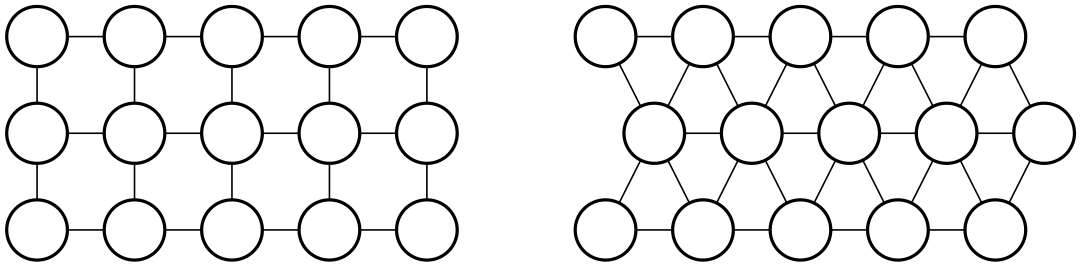
Input layer has as many neurons as attributes have the input patterns. Every neuron is connected to all neurons of the competition layer through weights (a weight exists

between each neuron of input layer and each neuron of competition layer). Every input pattern is transmitted from input layer to competition layer, where activation is calculated for each neuron. Input patterns must be numeric vectors, i.e., every vector component (attribute) must be a real number.

Competition layer usually has a two-dimensional structure. Neurons amount is a parameter that user has to define when network is going to be created, including rows and columns. Every neuron receives a signal from input layers and calculates its activation. Neuron with highest activation is the winner neuron and its weights are updated, and weights of neighbor neurons are updated too, using a learning law.

Neighborhood function is a function whose arguments are a neuron, a distance function and a maximum neighborhood limit and returns neurons that are neighbors of the specified neuron and its distances to it. Euclidean distance is usually used as distance function. Neighborhood is a very important concept related to Kohonen maps, because changes to a neuron imply changes to its neighbors makes this kind of network have very interesting properties. Neighborhood limit is dynamic: it decreases while network is trained. Decreasing can be linear, exponential…

Disposal of neurons in the map influences neighborhood. Most common types are rectangular and hexagonal:



Learning rate is other parameter to configure. It is a real number in range [0,1] that limits weights changing when they have to be updated (because weights belongs to winner neuron or to a neighbor neuron of winner neuron). As neighborhood limit, learning rate decreases while network is trained. Both parameters decrease imply the map will change a lot at beginning and it will change a few at the end.

Learning law applied is:

$$\Delta\mu_{ij} = \begin{cases} \dfrac{\alpha(t)}{\sigma(c_k, c_j)}\left(e_i(t) - \mu_{ij}(t)\right), \text{if } c_k \text{ is the winner neuron and } \sigma(c_k, c_j) \leq \theta(t) \\ 0, \text{in any other case} \end{cases}$$

where:

- μ_{ij} is the *i* weight of the *j* neuron.
- $\alpha \in [0,1]$ is the learning rate.
- $\sigma(c_k, c_j)$ is the neighborhood degree between *i* and *j* neurons. It is: $\sigma(c_k, c_j) = 1 + d(c_k, c_j)$ where $d(c_i, c_j)$ is the distance between *i* and *j* neurons. If chosen distance function is the euclidean distance:

$d(c_k, c_j) = \sqrt{\sum_{i=1}^{n}(\mu_{ik} - \mu_{ij})^2}$, n is the amount of attributes of patterns.

- e_i is the i attribute of the e pattern.
- $\theta \in \mathbb{R}^+ \geq 1$ is the neighborhood limit.

Training a Kohonen map could be summarized in the following steps:

1. Initialize weights randomly with small values.
2. Take an input pattern.
3. Calculate distance from pattern taken to every neuron.
4. Select the winner neuron, the nearest to the pattern.
5. Get neighbor neurons of winner and distances to it.
6. Apply the learning law.
7. Return to step 2, until every pattern have been taken.
8. Update the learning rate (α) and the neighborhood limit (θ).
9. If this is the last iteration, end. In other case, return to step 2.

Even though this type of neural network makes unsupervised machine learning, they could be used for supervised learning, if map is calibrated after the training.

Calibration is a process that assigns a class or a label to each neuron. To know what label assign to a neuron, it is checked what patterns of the training dataset are represented by that neuron and what labels they have. Most repeated label is assigned to the neuron.

It is important to notice that the learning method used for the training process is unsupervised and the map calibration is performed once completed, without having affected to the training. After performing this labeling, not all neurons have to have an assigned class. There may be "loose" neurons without patterns to represent and, therefore, a class cannot be assigned. Although it is possible that neurons do not represent any pattern, each pattern is represented by exactly one neuron.

If after calibrating the map training dataset were classified, every pattern would be classified by its nearest neuron and there would not be patterns without classifying. However, if test dataset were classified, it could contain patterns whose nearest neuron was not labeled during calibration and they would not be classified. In his case, these patterns would be counted as misses.

Supervised learning can be made in this way:

1. Configure map parameters and train it (this step does not take into account that training dataset is labeled).
2. Calibrate the map using training dataset labels.
3. Calculate hit rate of training and test datasets.
4. Return to step 1 if want to try with other maps.

[1][2]

7.3.3. Technical solution design

7.3.3.1. Design alternatives

When designing the application, it is important to note that the chosen programming language is Java, an object-oriented language.

For the arrangement of visual elements in the user interface, Java provides different templates or layouts. Layout used is "GridBagLayout" because, after comparing it with other in a web of Oracle, it seems to be one of the most permissive options to arrange items and to resize the application by specifying their behavior to do so. Moreover, there was not an exact idea from the beginning about how the application should be graphically (while giving necessary options), "GridBagLayout" has been chosen to use because of being one of the layouts most tolerant to changes. [11][12]

It organizes items as if space were a table or grid with as many rows and columns as chosen. In their cells, elements are placed. In addition, all rows do not need to have the same high or columns the same width, it is possible to make an item span multiple rows and / or columns if desired. This type of organization allows to resize the application window by the user and its items will resize too, if established. [13]

For the graphics, it was decided to use two Java libraries: "Graphics" and "Graphics2D". They provide all necessary operations to draw every network training process iteration, in an easy way, taking into account application requirements. It was not considered necessary the possibility of using other libraries, because application will display simple shapes that are readily available with mentioned Java libraries and whose way to use can be found in the API (Application Programming Interface) and other Oracle websites that teaches how to do some things in Java. [15]

If graphics were more complicated, for example, using many different geometric shapes, Cartesian axes, images and / or animations, three-dimensional objects, allowing the user to change the viewpoint, etc. possibility of using other libraries would have been taken into account.

For the graph part that shows error evolution related to training, it was decided to use "JMathPlot" library [16], which allows an easy way to do it. There are alternatives such as "JFreeChart" [17], with more options, but in this case it is not necessary to make more complex graphs, so using "JMathPlot" it is enough, saving space in the final application size without losing functionality.

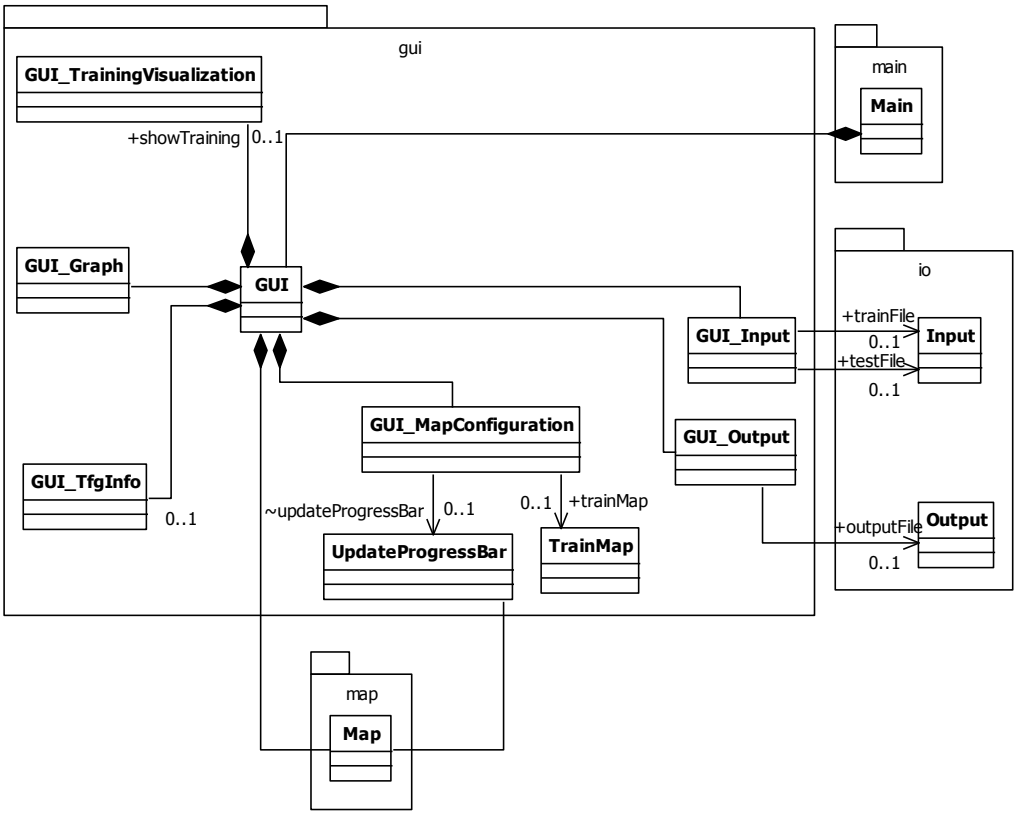
Graphic training is an essential part of the application. Because of that, it is important to ensure that always it is displayed rightly. Since the application needs to be displayed continuously on the screen and to perform training process calculations, using threads is necessary, to ensure there are not delays in user interface while network is being trained. There must also be communication between threads to be sure that everything is painted on the screen corresponds to a concrete process iteration. Otherwise, there could be a situation where what is shown on screen may not be consistent. It could occur, for example, an iteration number is displayed while position of some neurons and learning rate shown correspond to next iteration.

To avoid such situations, a *mutex* has been used, which enables proper coordination between different threads actions. It is a mutual exclusion mechanism (MUTual EXclusion) that avoids a thread to execute its critical section if another thread is already executing its one, and when completed will allow first to continue executing. Furthermore, it has been used to require that the graph was updated at least once per iteration, regardless of the management process performed by the operating system so that it can avoid there were iterations not shown.

Threads management could have been made with other mechanism, such as semaphores. A *mutex* was used because threads control was easy: a process train while other one paints the screen. If more than two processes had needed to access same data, maybe semaphores would have been more appropriate, since they can allow many threads to execute simultaneously.

7.3.3.2. Application design

Then, UML (Unified Modelling Language) class diagram of application is shown, only with classes and relations among them, but without attributes or operations.



The application source code is divided in some packages:

- main. It contains “Main” class. It only initiates the application, creating a “GUI” object.
- gui. It contains all classes related to GUI (Graphic User Interface). They allow right communication between a user and the application. The application, from a visual viewpoint, is divided in five parts:

    - Input. Class that shows this part is “GUI_Input”. It contains all items that allow the user to specify, if there are, input train and test files, or if the user prefers to use graphic input.
    - Map configuration. “GUI_Map_Configuration” class contains all items to allow the user to introduce map parameters.
    - Output. Its items are shown by class “GUI_Output”. It is a panel that allows the user to save different files with information about network training results. If user has made a training dataset with mouse clicks, he can save it.
    - Graphic training visualization. “GUI_TrainingVisualization” and “GUI_Graph” classes show how network is trained. Class “GUI_TrainingVisualization” allows to delete patterns and neurons, to change their colors, to adjust training speed and to delete the graph to allow to train with files. “GUI_Graph” class shows training graphically.
    - TFG Information. When the application starts, a second window appears in front of the main window, shown by “GUI_TfgInfo” class. It is a merely informative window, that allows a user to know that the application has been developed in a university to be a “Trabajo de Fin de Grado”. This window is only visible at beginning.

  “GUI” class contains the main window, which contains all these items. In addition, it contains the map that will be trained.
- io. It contains classes related to input and output operations. “Input” class reads input train and test files. “Output” class writes output files.
- map. It only contains “Map” class. It contains all attributes and operations necessary to define, to train and to calibrate a map.

7.3.4. Results and assessment

In these lines, it is described how application has been tested, including results. Next tests are defined from user viewpoint, taking into account what things could he want to do using the application and how he should do it.

In general, these are all things a user could want to do:

- Generate dataset graphically and train. This could be interesting for the user that is trying to understand intuitively how Kohonen maps work. He should:
    1. Press the “Crear entrada gráficamente” button.
    2. Click with left button mouse on the graph that has appeared where want to add a pattern.
    3. Configure map parameters.
    4. Press the “Entrenar” button.

- Generate dataset graphically, train and save the results. Useful if the user wants to learn intuitively how Kohonen maps work with more concrete data. In this case, the user should:
    1. Press the “Crear entrada gráficamente” button.
    2. Click with left button mouse on the graph that has appeared where want to add a pattern.

3. Configure map parameters.
4. Press the “Entrenar” button.
5. Depending on the results that want to save, press the appropriate button.
6. Select a path and a name for the file that will be written.
7. Return to step 5 if want to save more results.

- Train without graph using training and test files and save the results. An useful option for a user who has understood how Kohonen maps work and wants to apply them using files. The user should:
    1. Press the “Seleccionar archivo de entrenamiento” button.
    2. Use the explorer to select the training file.
    3. Press the “Seleccionar archivo de test” button.
    4. Use the explorer to select the test file.
    5. Configure map parameters.
    6. Press the “Entrenar” button.
    7. Depending on the results that want to save, press the appropriate button.
    8. Select a path and a name for the file that will be written.
    9. Return to step 7 if want to save more results.

- Train with graph using training and test files and save the results. This is an interesting option for a user who want to learn about Kohonen maps from concrete data, without losing the possibility of seeing the training process. The user should:
    1. Press the “Seleccionar archivo de entrenamiento” button.
    2. Use the explorer to select the training file.
    3. Press the “Seleccionar archivo de test” button.
    4. Use the explorer to select the test file.
    5. Activate the “Mostrar entrenamiento gráficamente” checkbox.
    6. Configure map parameters.
    7. Press the “Entrenar” button.
    8. Depending on the results that want to save, press the appropriate button.
    9. Select a path and a name for the file that will be written.
    10. Return to step 8 if want to save more results.

Instead of following those steps strictly, some of them can interchange order and there are more options for the user, such as initializing the neurons manually, change the colors of patterns, neurons and / or background, delete patterns and / or neurons, adjust the training speed, show / hide neurons coordinates, etc.

Besides having tested evaluating the cases described above after finishing the implementation, during development and after finishing it many more tests were done in order to check that application requirements were satisfied.

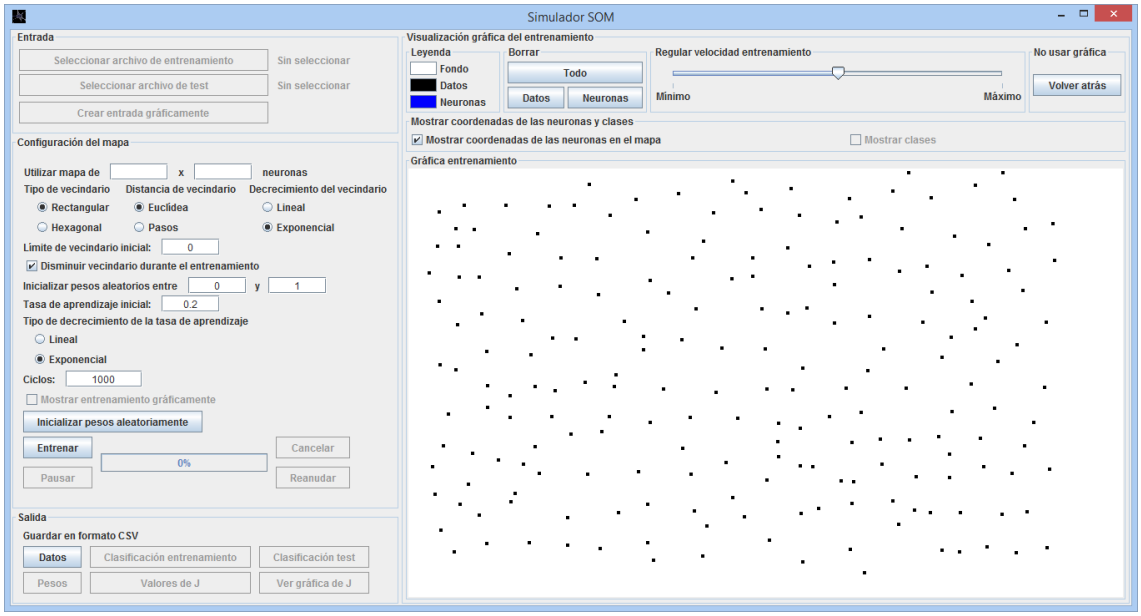
Many tests should be done after implementing code related to them, because it was important to know that something basic worked well before carrying over implementation. Later, it could be checked again with different tests if necessary.

Finally, it is possible to say that results of all tests have been satisfactory, ensuring the application quality. One of the consequences is that it has been possible to develop all requirements, even though they were optional.

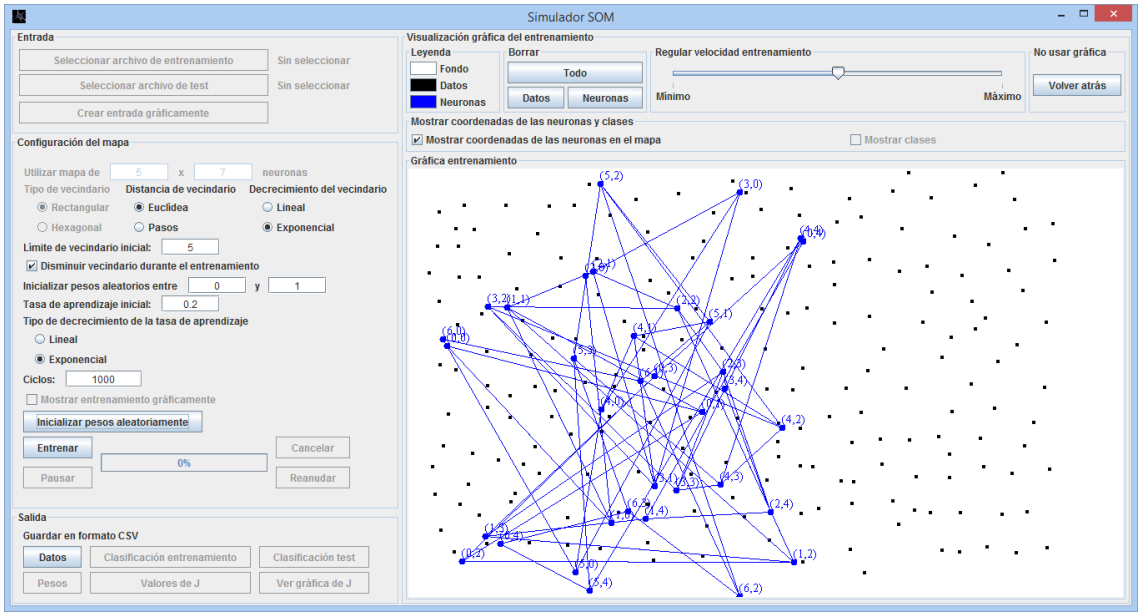
7.3.4.1. Examples

7.3.4.1.1. Example 1: uniform data distribution

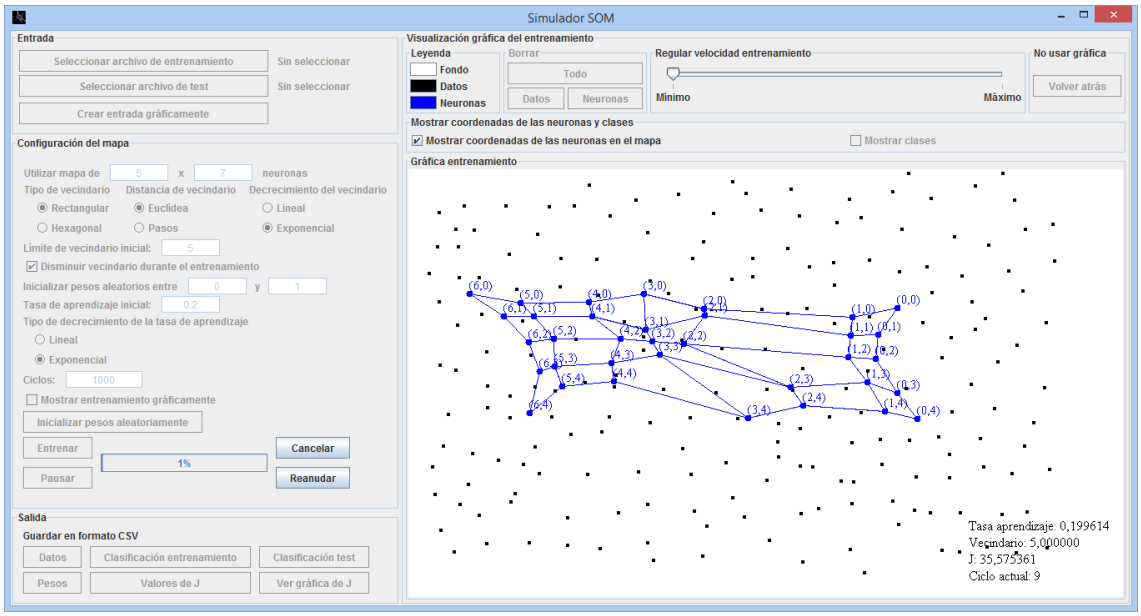
With a random uniform data distribution



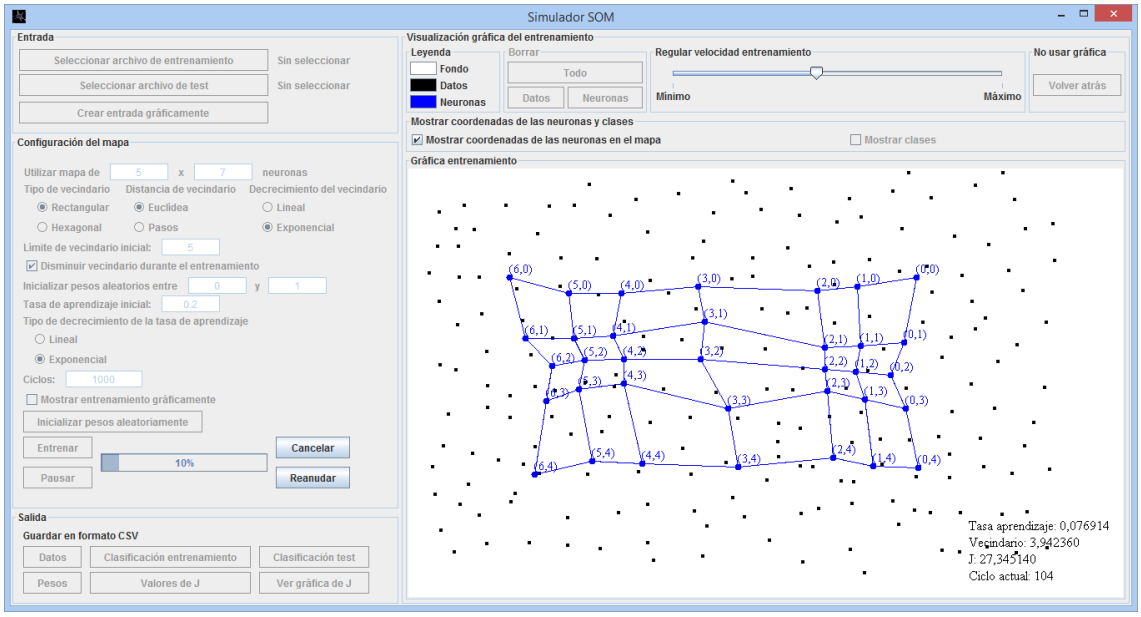
Patterns have been added uniformly and randomly, with mouse clicks.



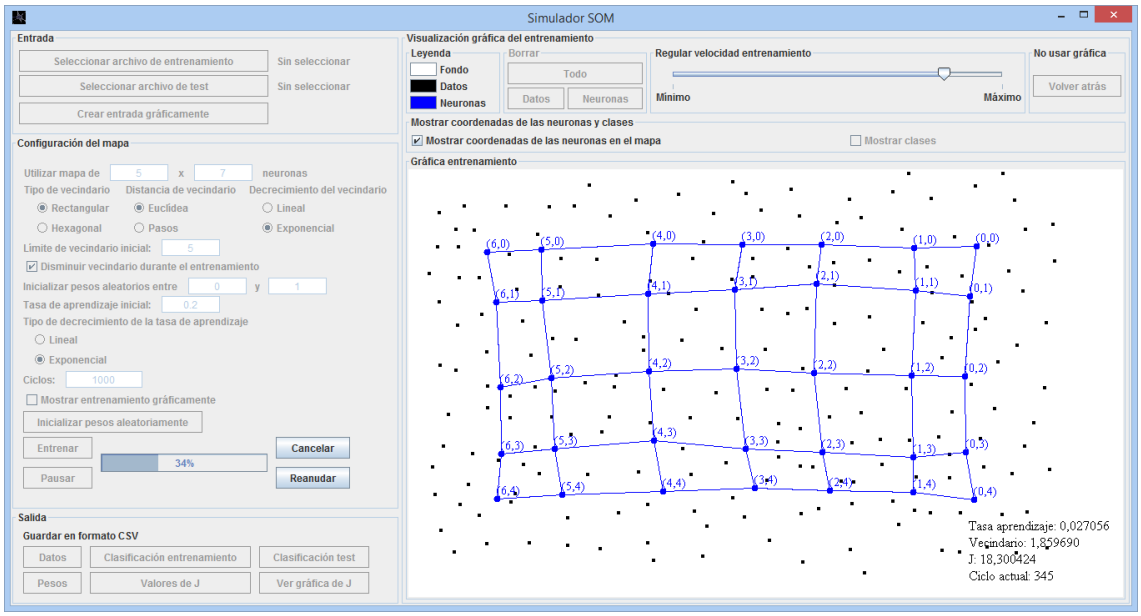
A 5×7 map is configured and initialized randomly. Rest of parameters has their default value, except initial neighborhood limit, that has been increased.



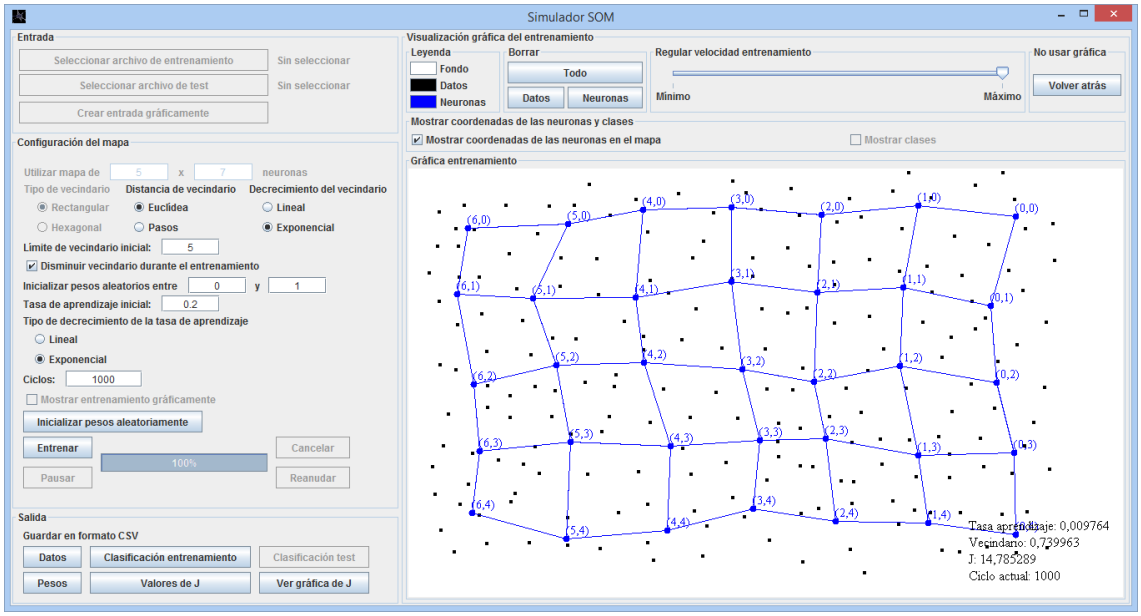
It is possible to see how in the ninth iteration map has changed a lot from the random initialization. This happened because, in the first iterations, limit neighborhood and learning rate are higher, allowing more abrupt changes. The position of the map is close to the central position of the data set.



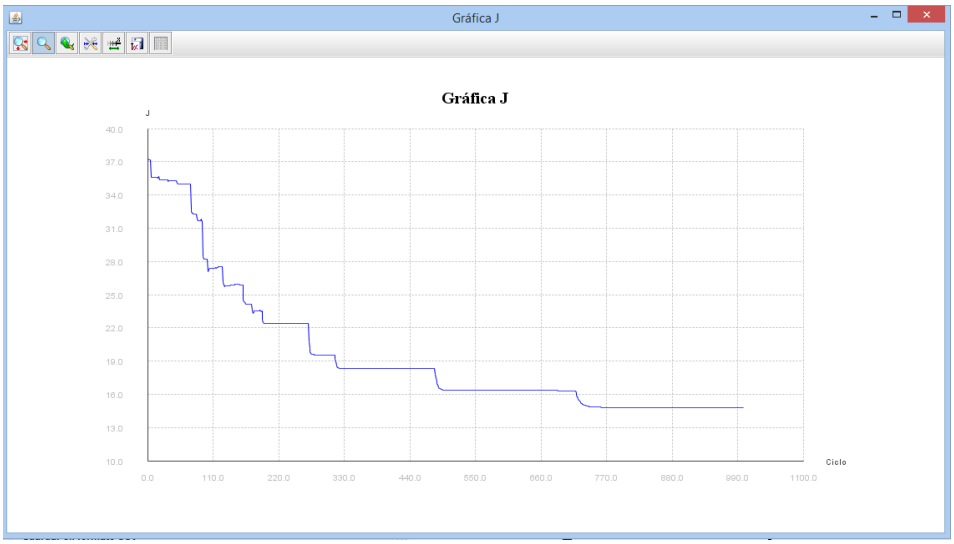
In the 104 iteration, neighborhood limit has decreased since the ninth iteration and neurons have separated.



Iteration 345: neurons have separated more and their distribution in the input space is quite uniform.



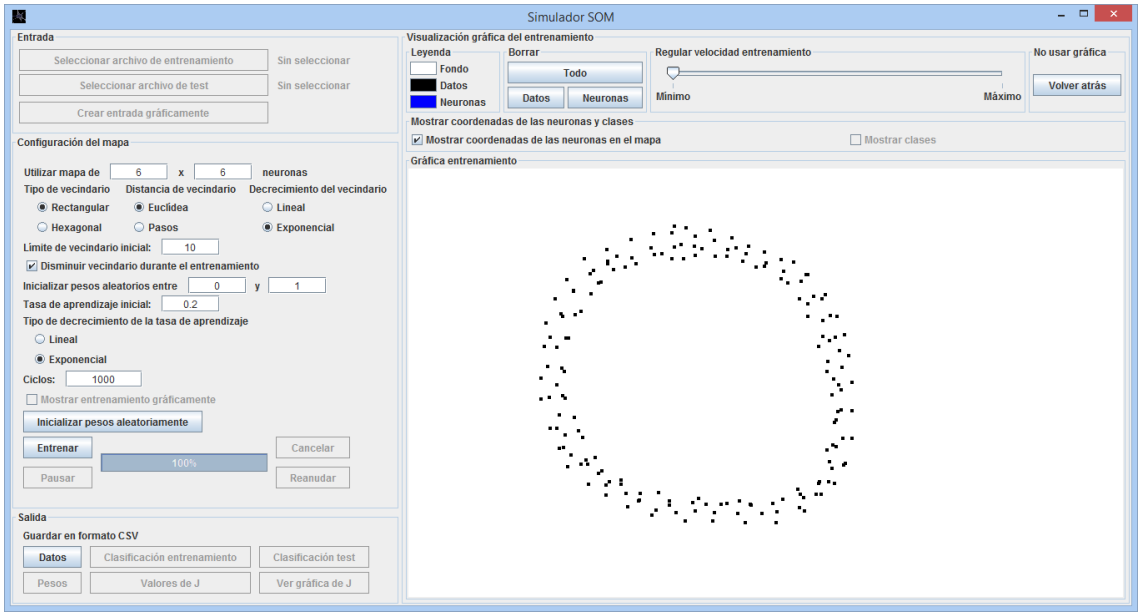
Finished training: it is possible to see that neurons are disposed like a grid, because of having used a uniform distribution.

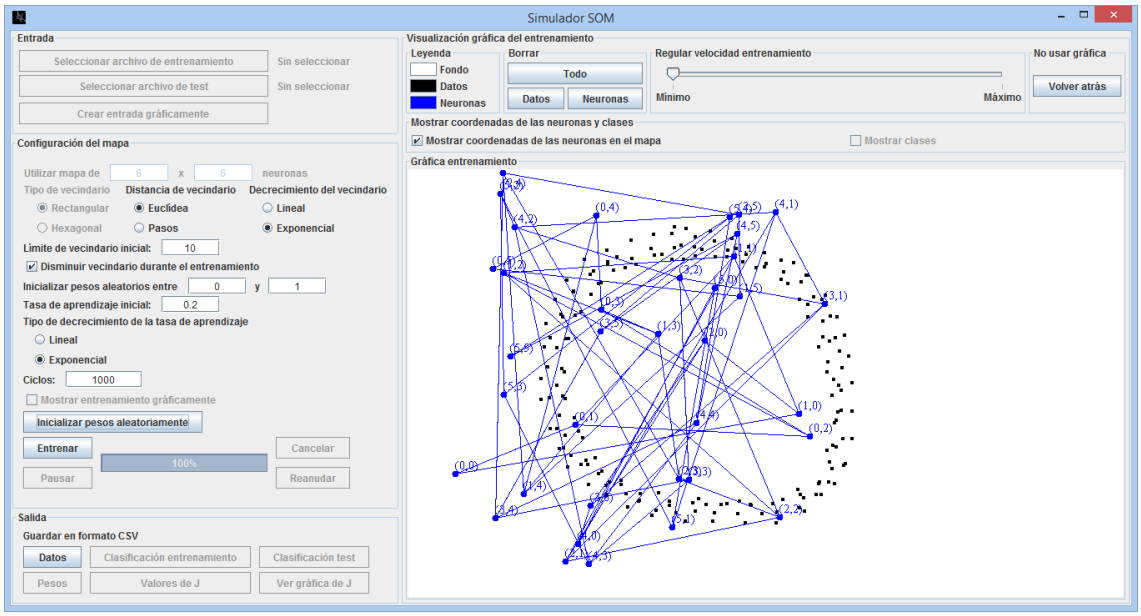


Error evolution: error made has decreased during training. There are some "peaks", probably because when decreasing neighborhood limit implies changing the amount of neighbors that a neuron has, changes are more abrupt.

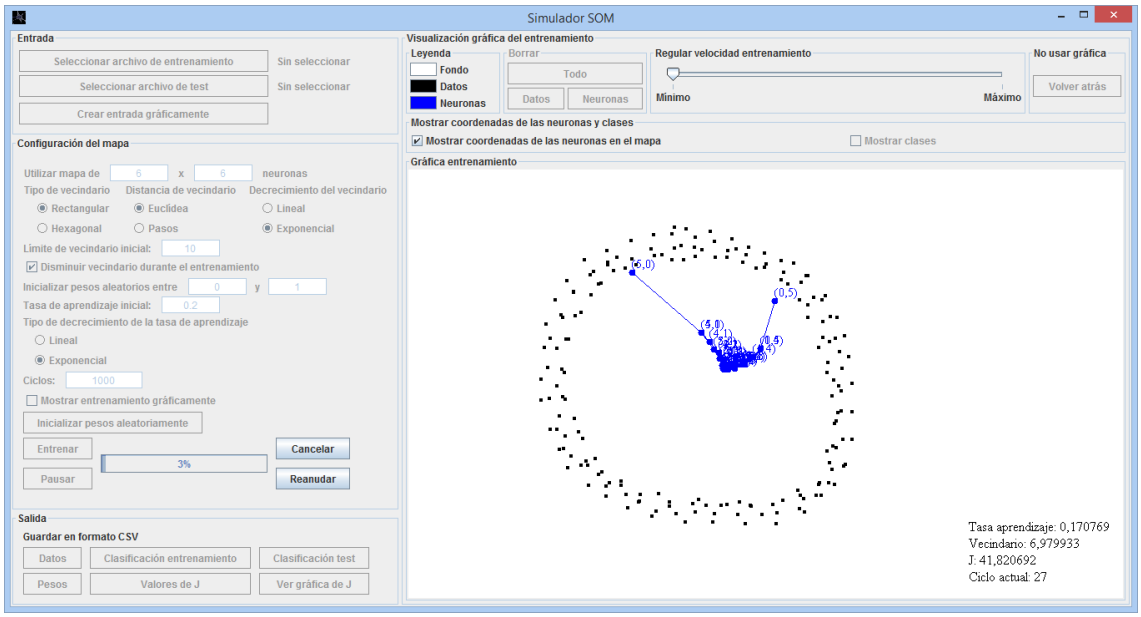
7.3.4.1.2. Example 2: circular data distribution

In the next example, data distribution is circular:

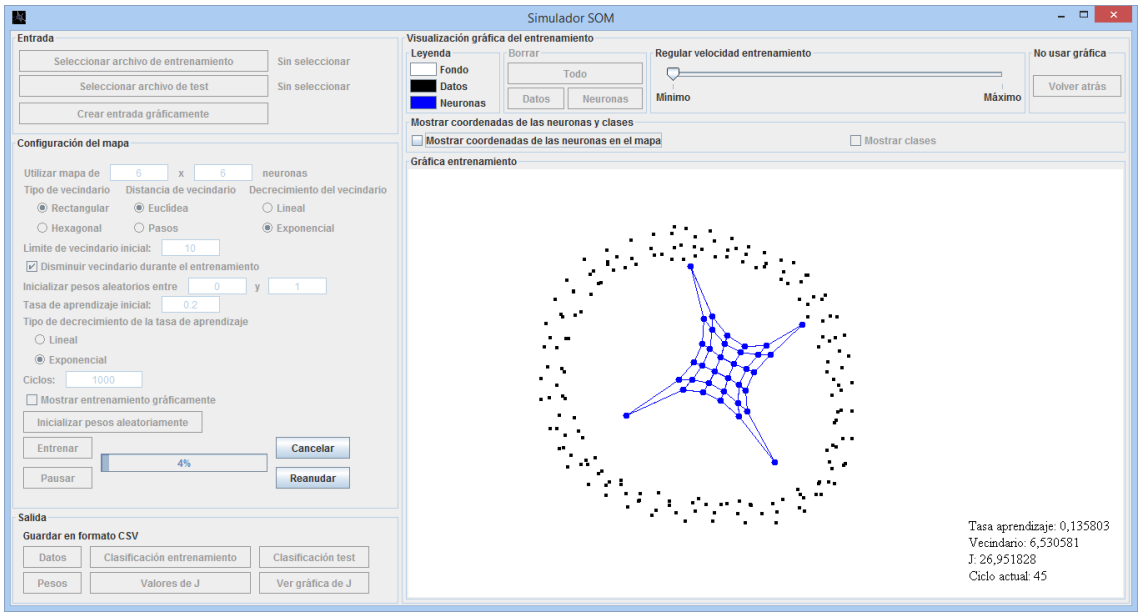




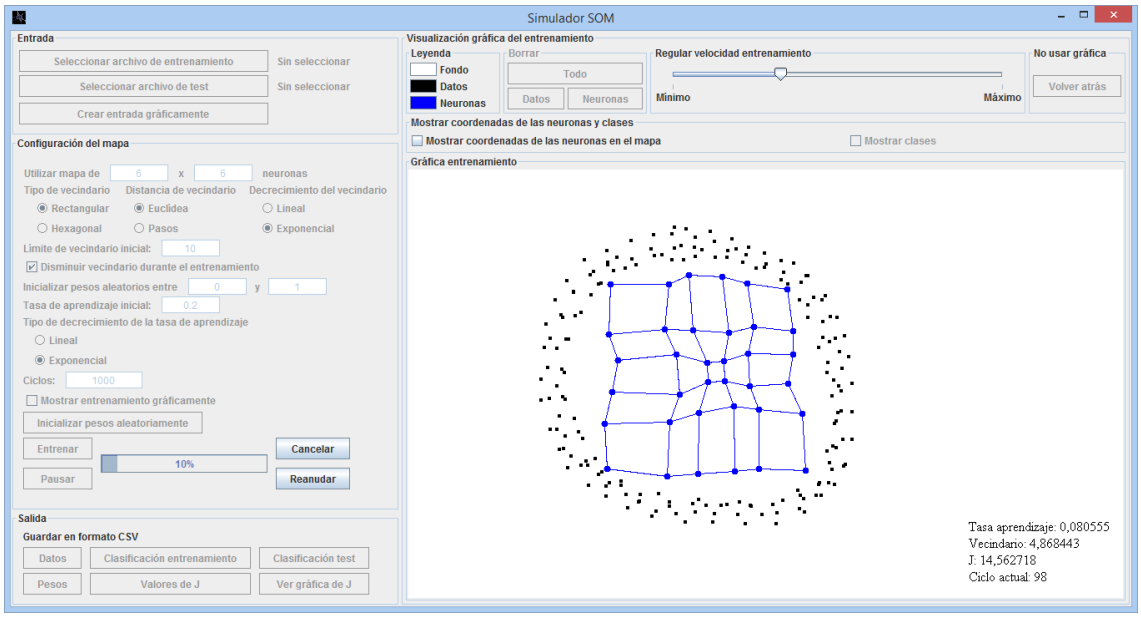
A 6×6 map is initialized randomly. Initial neighborhood limit is set to 10.



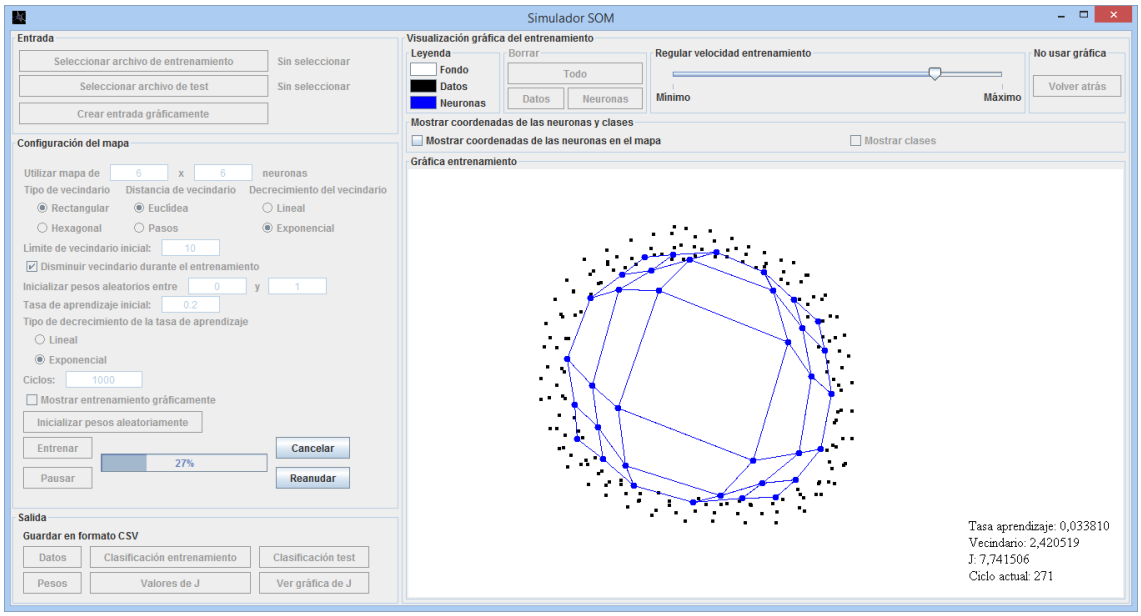
Iteration 27: map has changed a lot from random initialization again. Neuron set is close to patterns center.



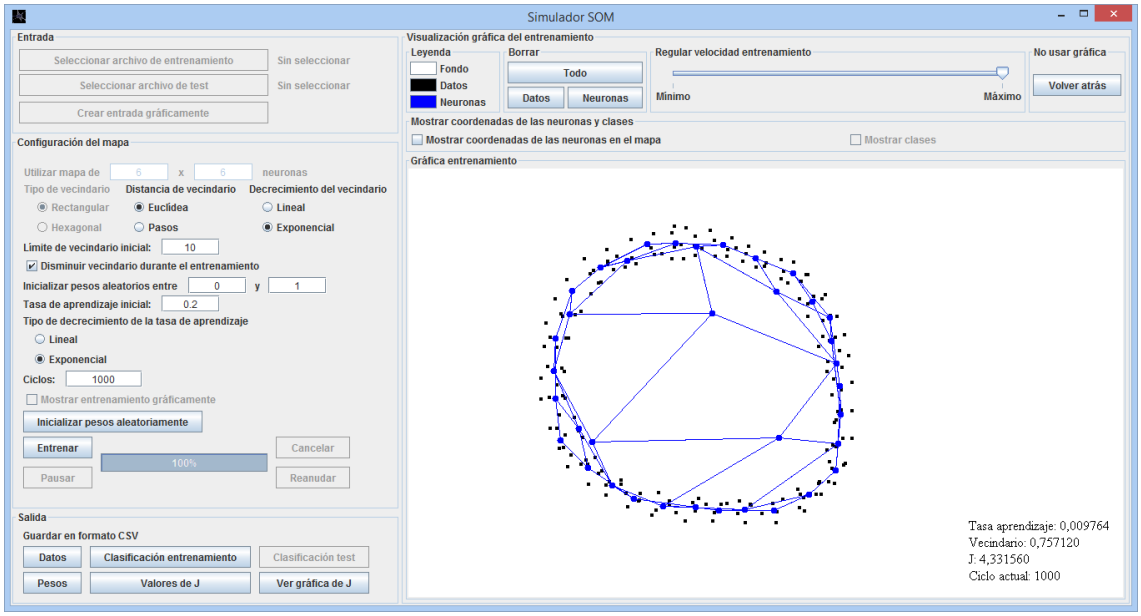
Iteration 45: neurons have separated a few.



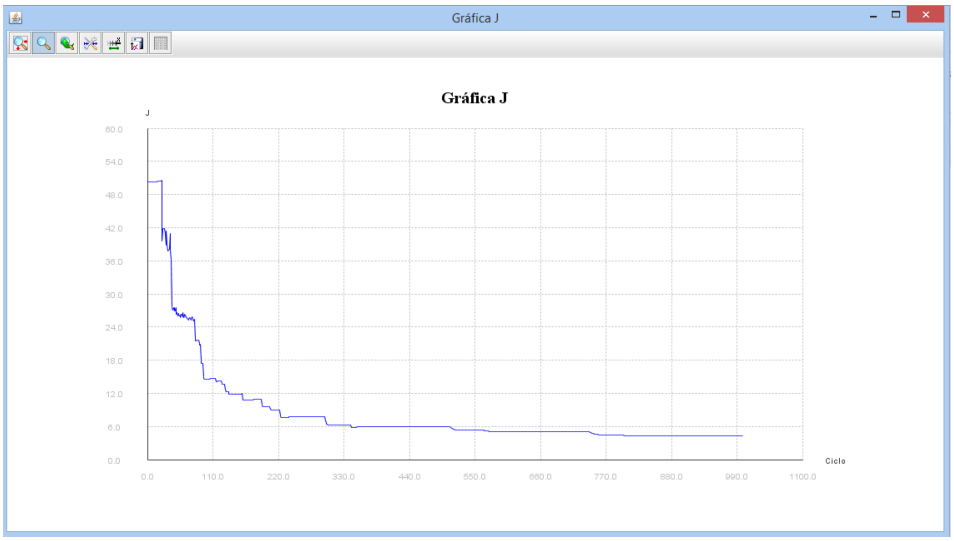
Iteration 98: neurons have separated more.



Iteration 271: neurons are adjusting to circular shape, due to neighborhood limit has decreased.



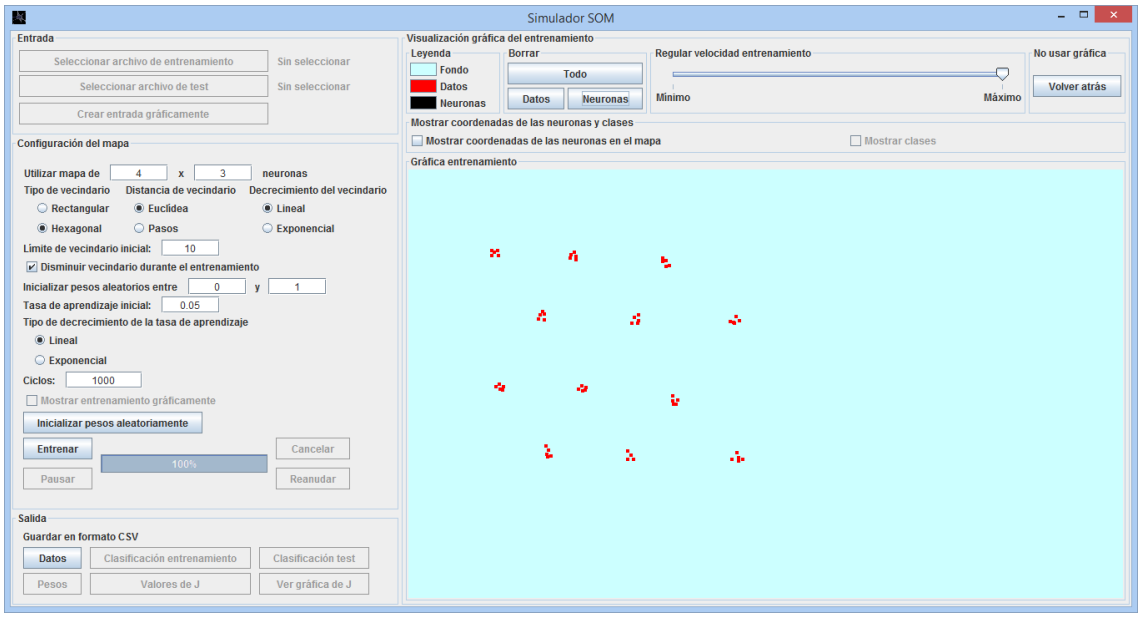
Finished training: neurons have adjusted to circular shape. It is possible to see two “loose” neurons, which would not represent any pattern. They are “loose” because other neurons have moved them.



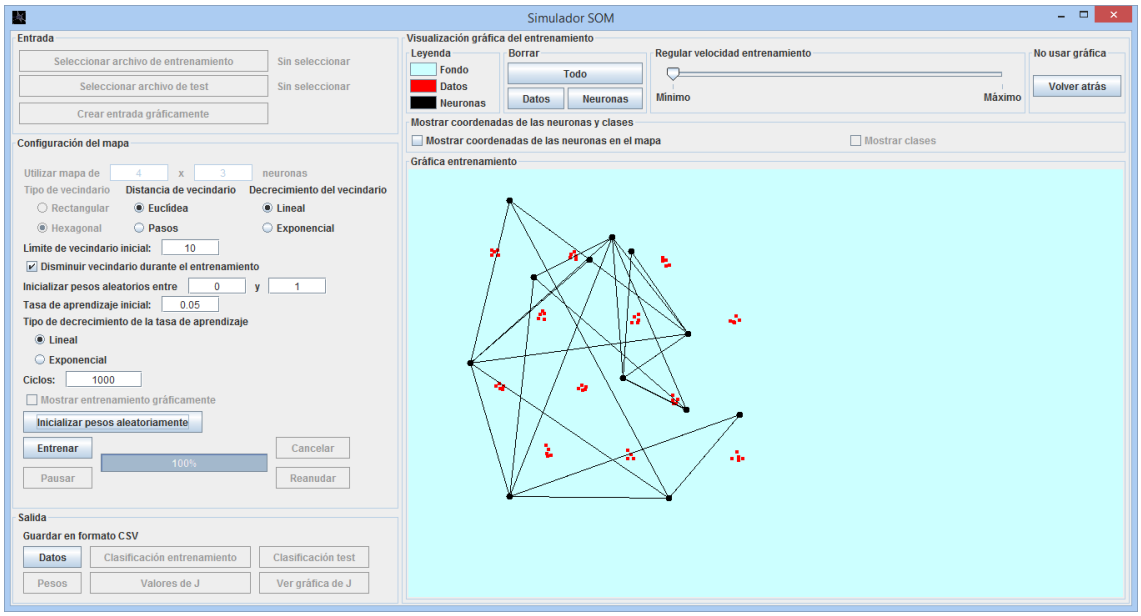
Error evolution: decrease is more abrupt at the beginning, when map changes are more abrupt.

7.3.4.1.3. Example 3: data disposed intentionally

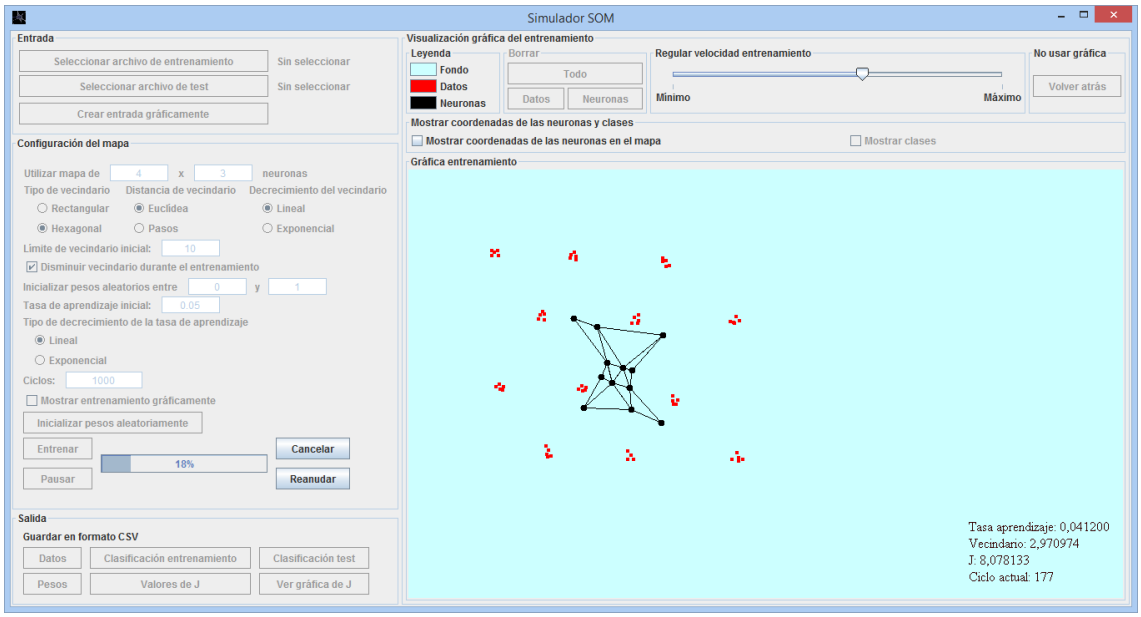
In the next example, imagining how neurons will be disposed after the training is easy:



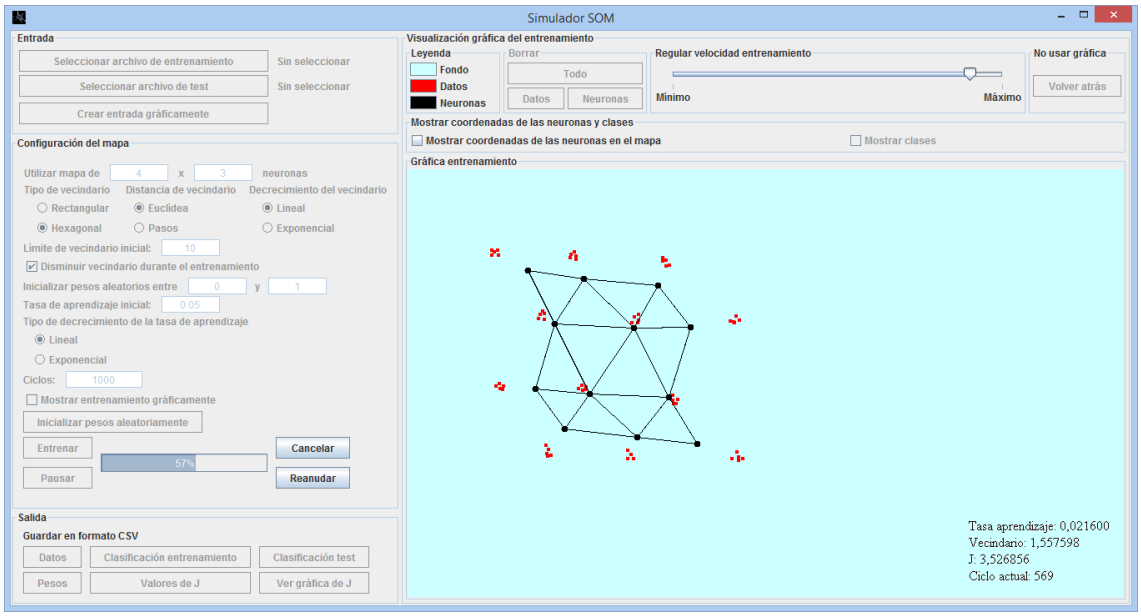
Patterns are divides in small groups that form four rows and three columns and they are disposed like a triangular grid. A 4×3 map with hexagonal structure will be used. This time, neighborhood limit and learning rate decrease will be linear. Initial learning rate value is 0.05. Initial neighborhood limited is 10. Patterns, neurons and background color have been changed.



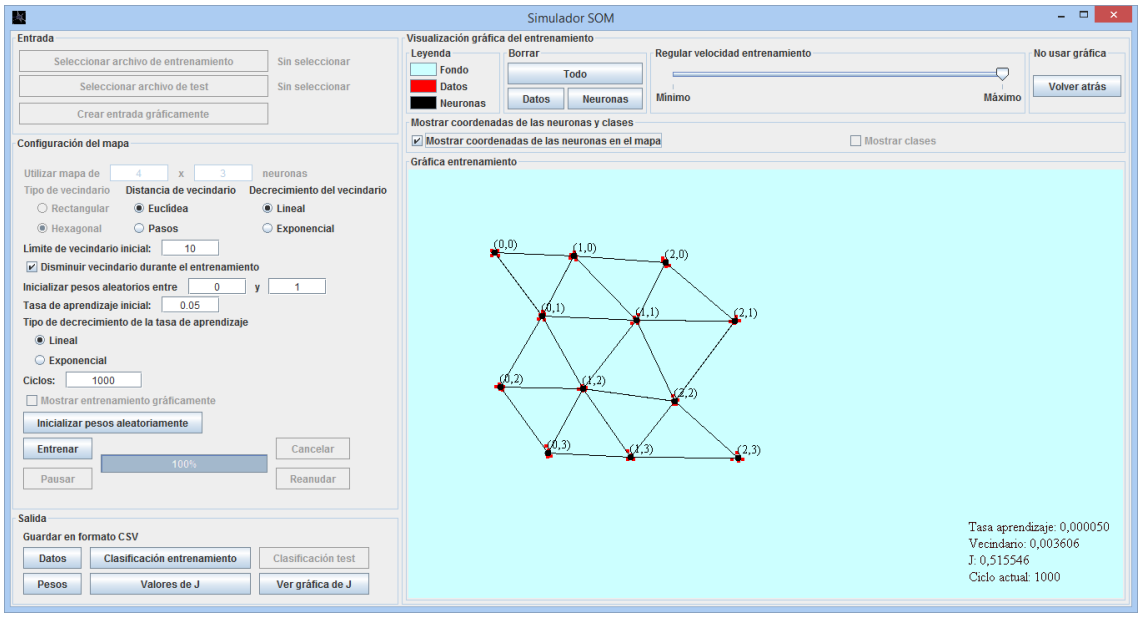
Neuron weights are initialized randomly.



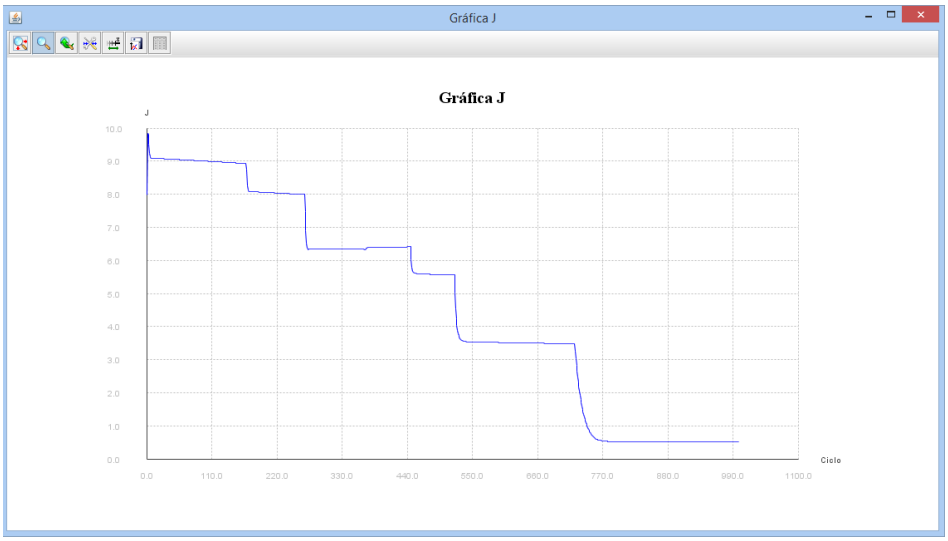
Iteration 177: neurons have changed a lot from the beginning. They are very near among them.



Iteration 569: neurons are adjusting their positions to the patterns.

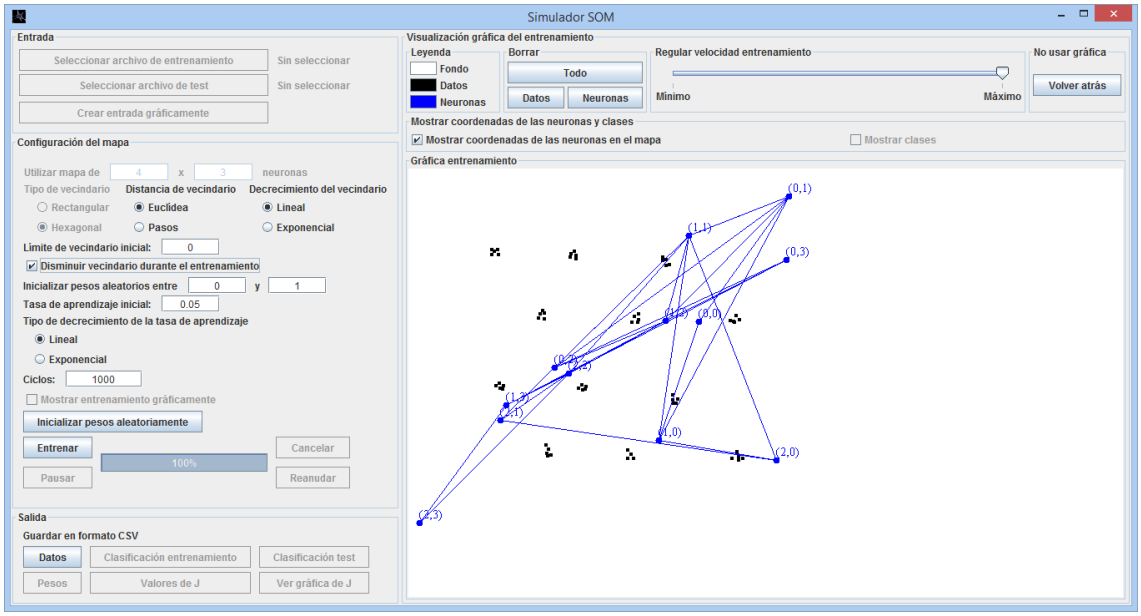


Finished training: each neuron is placed on a small group of patterns.

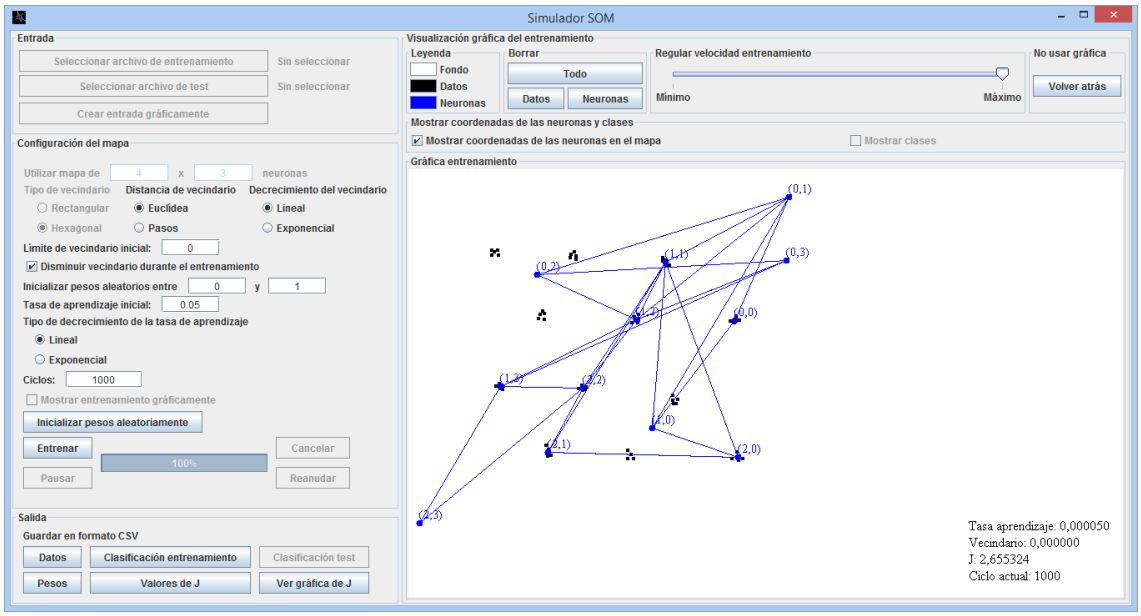


Error evolution: error has not decreased a lot at the beginning and a few at the end, because learning rate and neighborhood limit decrease were linear.

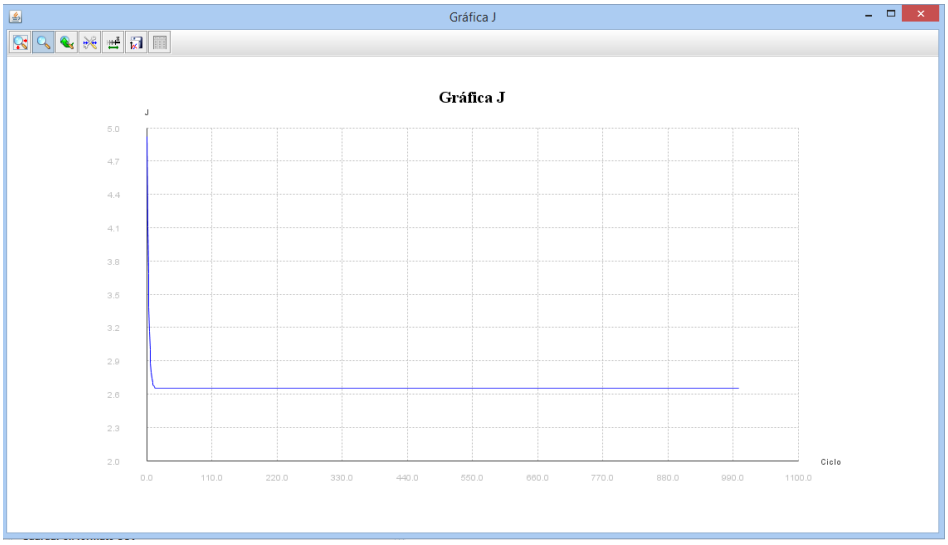
7.3.4.1.4. Example 4: without taking into account neighborhood



Neurons are initialized randomly and initial neighborhood limit is set to 0.

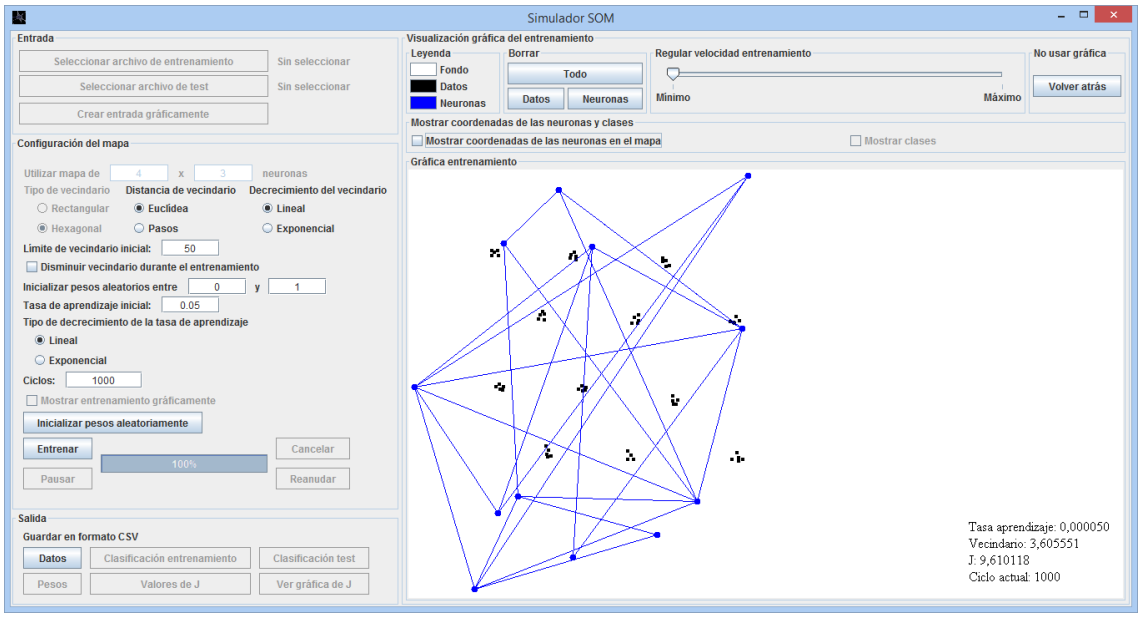


Finished training: after 1000 iterations, neurons are distributed in a similar way that after the random initialization.

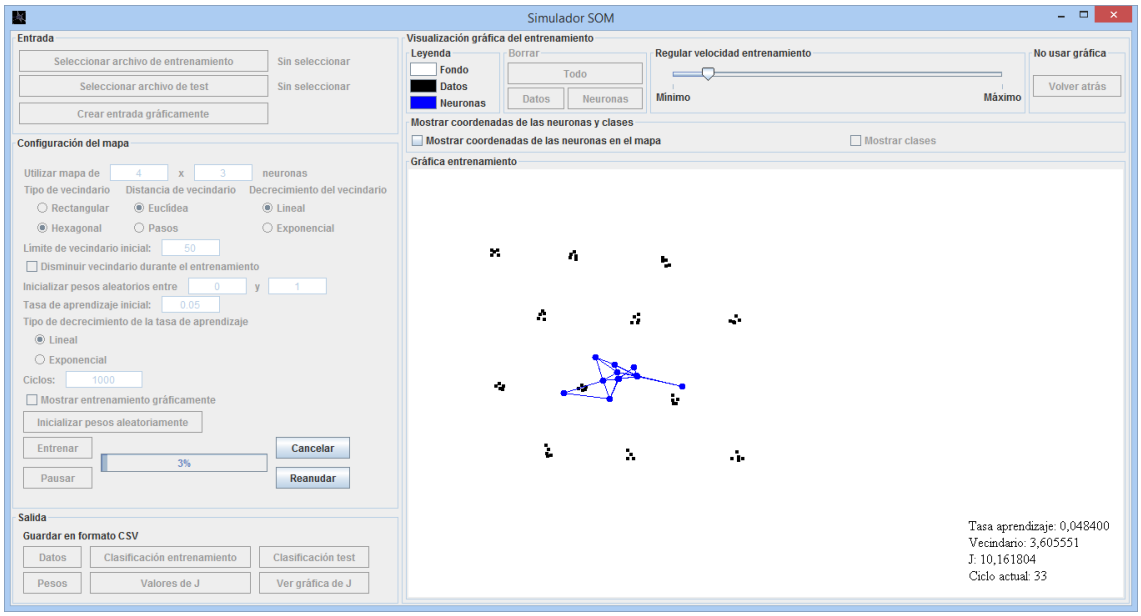


Error evolution: error is constant almost all the training.

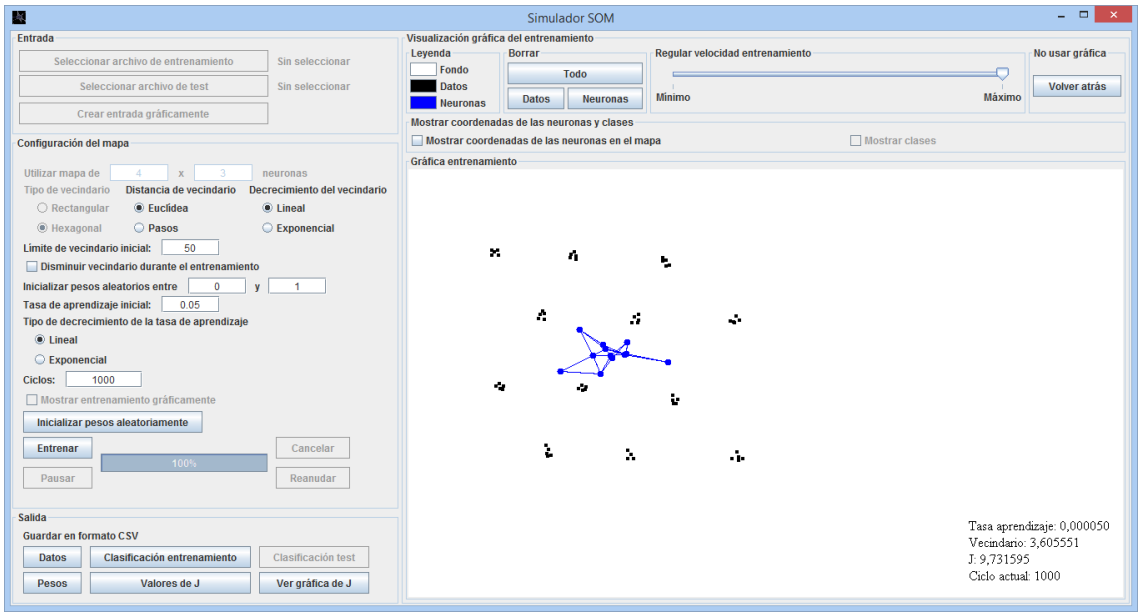
7.3.4.1.5. Example 5: without decreasing neighborhood limit



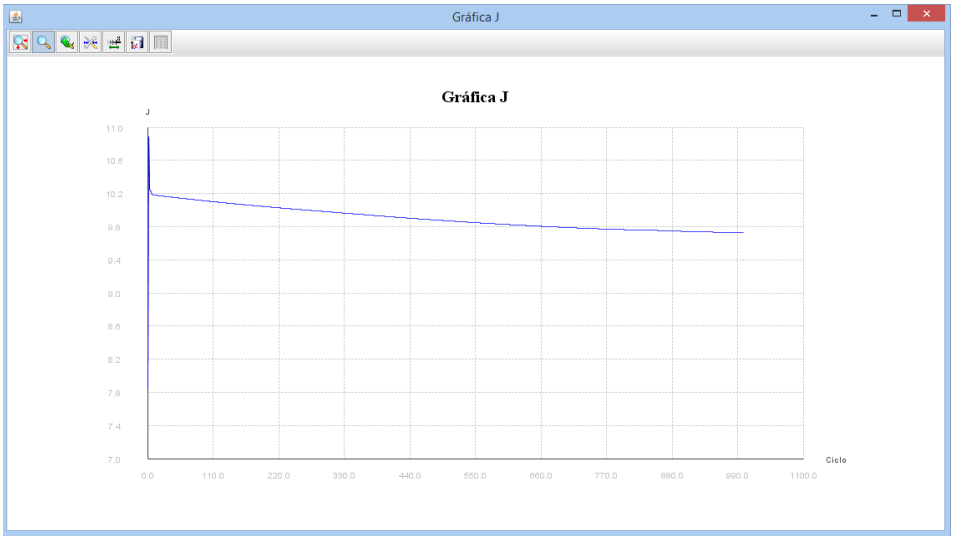
Neurons are initialized randomly.



Iteration 33: neurons are closed and centered.



Finished training: neuron still are closed and centered. They have not been able to separate because neighborhood limit has been high during all the training process. When a neuron tried to change its position, it did it changing positions of the rest of neurons.



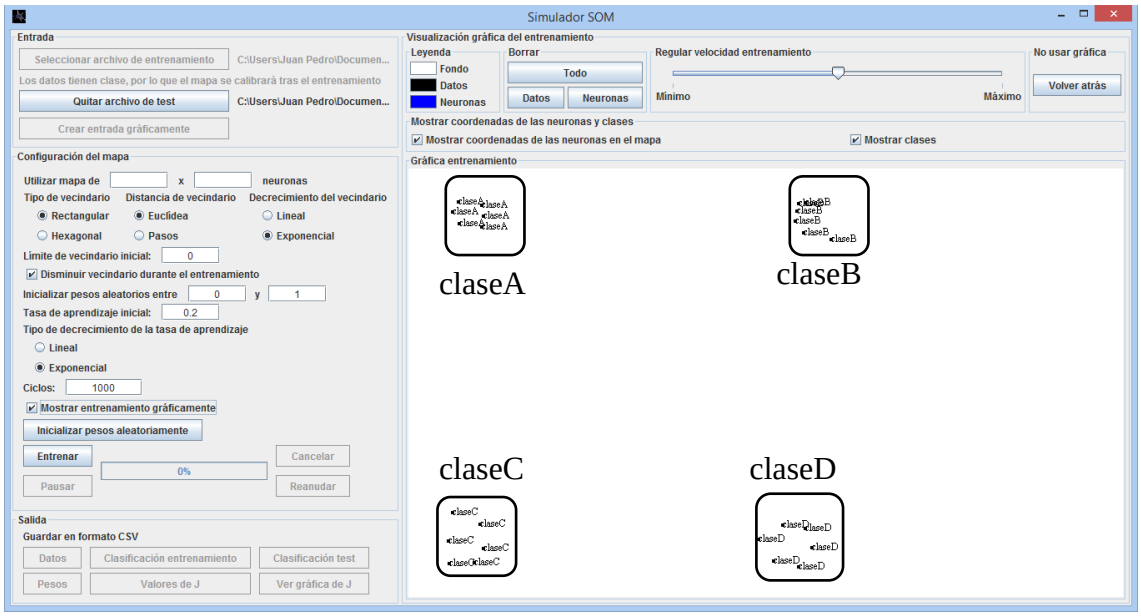
Error evolution: error decrease has been very low. It has not been minimized, as desired.

From a theoretical perspective, these examples have proved importance of neighborhood in Kohonen maps and they allow to get an intuitive idea about how they work. In addition, they have been useful to understand how some parameters influence.

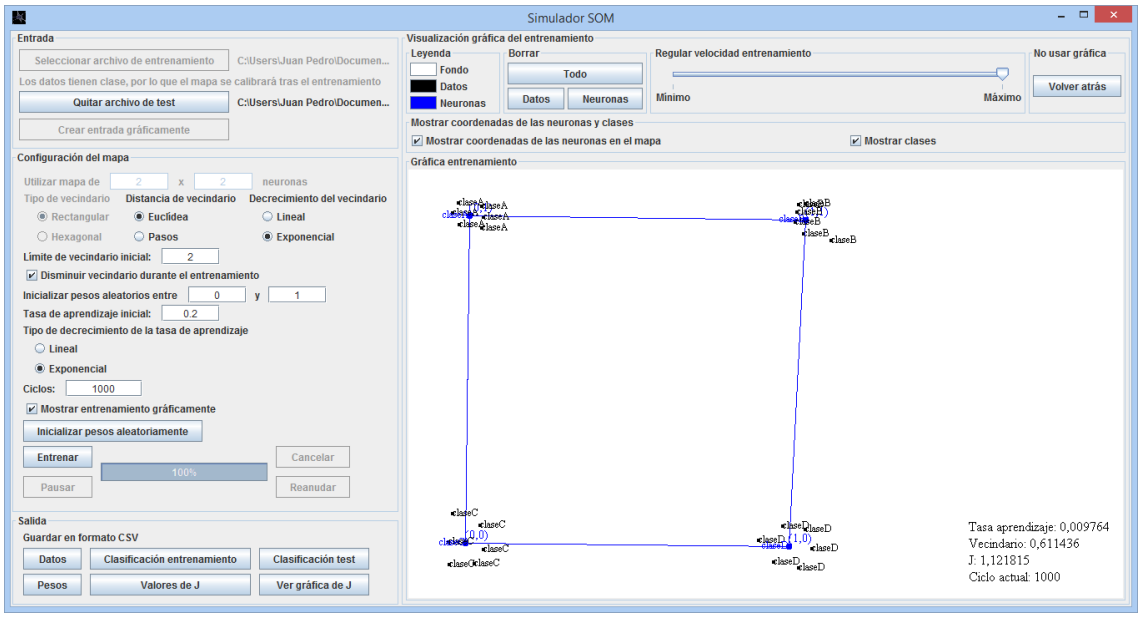
From a project perspective, these examples show that some objectives have been reached. It has been shown the ease when adding patterns and how is possible to see the training process and understanding what is happening. Training can be paused, cancelled or resumed when paused. When training has finished, a graphic to show error evolution may be generated.

7.3.4.1.6. Example 6: using input files and calibrating

Training input file dataset distribution is:



There are four different labeled groups. This is the result after training with a 2×2 map:



Next to each neuron, labeled assigned after calibration appears. Those labels are used to classify training and test files and they are exposed in the file that contains neurons weights.

This last example shows how the application is able to calibrate the map after training automatically, without being more complicated for the user.

7.1. Conclusion

7.1.1. Conclusion

This project had two main objectives: making easier the theoretical learning about Kohonen self-organizing maps and allowing to train them with real data.

First objective comes from necessity of teaching pupils who do not know what Kohonen maps are its behavior in a most intuitive possible way. Thus, it will be possible to make them think of acquired ideas to understand better the mathematical equations that define them meaning and influence of their parameters.

This objective is considered achieved, because it has been showed that next things are possible: add patterns on a graph simply with mouse clicks, configure parameters and training seeing what happens. Moreover, users have an option to save generated patterns to load them later in the application.

It is a very important objective, as, although there are other tools that allow to see how a map is trained, generally they are not so permissive as they could with input data and parameters configuration.

Second objective was to make the application could solve real problems, whose data is in files. That implied the application was able to read files with a specific format. It also implied the application was able to train maps, but already it was, because of having achieved previous objective. But patterns added with mouse clicks are not labeled. To achieve completely this objective, it was very interesting to add the possibility of calibrating the map if patterns were labeled. Calibration is the process of assigning a class (or not) to each neuron, taking into account patterns it represent when training has been finished. Training is always unsupervised.

To achieve the first objective, there was not an application with all the required characteristics. However, to achieve the second objective there are some applications, like “SOM_PAK”, the most representative.

This objective is considered to be satisfied too, as the application can train reading data files, calibrate map if data is labeled and use a test file. Moreover, the application can save a detailed file that identifies and specifies every neuron weights of the map.

Personally, this project has allowed me to experiment the process of making a complete a complete software project, instead of exercises that only takes some parts. As the only person in the project except me was mi tutor, making different parts of the project work well together was relatively easy, even with changes. However, large software projects with teams with more people, where some people make some things while other people make other things, it is essential they coordinate and communicate well.

7.1.2. Future trends

Even though work made is considered completed and achieved objectives, always it is possible add improvements.

When an automatic learning algorithm is use, data used usually is not in a desired way to use directly. Often, it is necessary to make preprocessing. One of the common tasks is normalization. This process involves changing the values of all attributes so that they are in the range [0,1]. This change should take into account the maximum and the minimum values that each attribute can take. Thus, although there are attributes which can take values in a very wide range, from small to very high values, while others take values in a much smaller range, normalization ensures that all are taken into account similarly while learning. Without normalization, differences between them would influence the training.

Another data preprocessing task very common when working with artificial neural networks is to assign a numeric value to attributes with nominal value. A neural network cannot accept as input attributes with nominal values, all must be real numbers. So when an attribute can take nominal values, it is transformed to real numbers. Also, as normalize data is usually used, this transformation is often done to values between 0 and 1, as sparingly as possible (or sometimes they are coded using several attributes).

The application could automatically perform these tasks would be interesting and could be made almost without being less intuitive, despite offering more functionality. There are other features that could also be interesting, such as the feature extraction using methods like PCA (Principal Component Analysis) or CFS (Correlation Feature Selection) or the possibility of providing some visual information from the data even if they have more than two dimensions. However, they are further away from the main objectives of the project.

Another feature that could be considered is to allow change patterns size and neurons size on screen as if the application is on a screen or projector that will be seen by several people at different distances, it could look better. It could also be added an option to save the map with a format that allows the application to load it later.

8. Referencias

[1] Isasi, P., Galván, I.M., *Redes de Neuronas Artificiales. Un enfoque práctico.* Pearson 2004.

[2] Valls, J. M., Galván, I. M. (15/09/2011), *Material de Clase. Tema 5* [consulta: 10-09-2014], desde el sitio Web de OCW - UC3M: Disponible en: http://ocw.uc3m.es/ingenieria-informatica/redes-de-neuronas-artificiales/transparencias/material-de-clase.-tema-5.

[3] Kohonen, Teuvo *et al. Chapter 8. Self-organizing map*. Biennial Report 2002-2003. Laboratory of Computer and Information Science. Neural Networks Research Centre. Helsinki University of Technology. P.O. Box 5400. FI-02015 HUT, Finland. K. Puolamäki and L. Koivisto, editors. Otaniemi, febrero de 2004. [consulta: 10-09-2014]. Disponible en http://cis.legacy.ics.tkk.fi/research/reports/biennial02-03/cis-biennial-report-2002-2003-8.pdf.

[4] *Teuvo Kohonen, Dr. Eng., Emeritus Professor of the Academy of Finland; Academician.* Neural Networks Research Centre, Laboratory of Computer and Information Science. [consulta: 10-09-2014]. Disponible en http://users.ics.aalto.fi/teuvo/index.shtml.

[5] Kohonen, Teuvo *et al. SOM_PAK: The Self-Organizing-Map Program Package.* SOM Programming Team of the Helsinki University of Technology. Otaniemi, enero de 1996. [consulta: 10-09-2014]. Disponible en http://hackbbs.org/article/reds/form_de_base/SBI/labo5/som_pak.pdf.

[6] *SOM_PAK AND LVQ_PAK.* Neural Networks Research Centre. Laboratory of Computer and Information Science. Helsinki University of Technology. [consulta: 13-09-2014]. Disponible en: http://www.cis.hut.fi/research/som-research/nnrc-programs.shtml.

[7] Kistler, T., Franz, M. *A Tree-Based Alternative to Java Byte-Codes.* Departamento de Información y Ciencias de la Computación de la Universidad de California, Irvine [consulta: 22-08-2014]. Disponible en ftp://158.130.67.137/pub/cis700/public_html/papers/Kistler96.pdf.

[8] Oracle, *Conozca más sobre la tecnología Java.* [consulta: 22-08-2014]. Disponible en https://www.java.com/es/about/.

[9] Oracle, *The history of Java Technology*. [consulta: 22-08-2014]. Disponible en http://www.oracle.com/technetwork/java/javase/overview/javahistory-index-198355.html.

[10] Oracle, *How Will Java Technology Change My Life?* [consulta: 13-09-2014]. Disponible en: http://docs.oracle.com/javase/tutorial/getStarted/intro/changemylife.html

[11] Oracle, *A Visual Guide to Layout Managers* [consulta: 29-08-2014]. Disponible en http://docs.oracle.com/javase/tutorial/uiswing/layout/visual.html.

[12] Oracle, *Using Layout Managers* [consulta: 29-08-2014]. Disponible en http://docs.oracle.com/javase/tutorial/uiswing/layout/using.html.

[13] Oracle, *How to use GridBagLayout* [consulta: 29-08-2014]. Disponible en http://docs.oracle.com/javase/tutorial/uiswing/layout/gridbag.html.

[14] Oracle, *Solving Common Layout Problems* [consulta: 29-08-2014]. Disponible en http://docs.oracle.com/javase/tutorial/uiswing/layout/problems.html.

[15] Oracle, *The Java™ Tutorials. Trail: 2D Graphics*. [consulta: 29-08-2014]. Disponible en: http://docs.oracle.com/javase/tutorial/2d/.

[16] µ-Labs. *JMathPlot*. [consulta: 10-09-2014]. Disponible en https://sites.google.com/site/mulabsltd/products/jmathplot.

[17] *Welcome To JFreeChart!*. [consulta: 10-09-2014]. Disponible en: http://www.jfree.org/jfreechart/.

[18] Maximilian Bügler, *KOHONENPOWER v0.91*. 2008. [consulta: 10-09-2014]. Disponible en: http://www.maxbuegler.eu/kohonen/.

[19] España. Jefatura del Estado. *Ley Orgánica 15/1999, de 13 de diciembre, de Protección de Datos de Carácter Personal*. Última modificación: 5 de marzo de 2011. [consulta: 12-09-2014]. Disponible en: http://www.boe.es/buscar/pdf/1999/BOE-A-1999-23750-consolidado.pdf.

[20] The Eclipse Foundation, *Eclipse Public License - v 1.0*. [consulta: 12-09-2014]. Disponible en: https://www.eclipse.org/org/documents/epl-v10.php.

[21] Microsoft, *MICROSOFT SOFTWARE LICENSE AGREEMENT WITH COMPUTER MANUFACTURER OR SOFTWARE INSTALLER, OR MICROSOFT*. [consulta: 12-09-2014]. Disponible en: http://download.microsoft.com/download/7/2/1/72103B1C-DE58-478C-BECF-4D846C2B4A15/UseTerms_Retail_Windows_8.1_N_update_(to_Windows_8_preinstalled_on_your_computer)_English.pdf.

[22] Microsoft, *Contrato de Licencia Del Software de Microsoft*. [consulta: 13-09-2014]. Disponible en: http://office.microsoft.com/es-es/products/contrato-de-licencia-del-software-de-microsoft-FX103576343.aspx.

[23] España. Ministerio de Cultura. *Real Decreto Legislativo 1/1996, de 12 de abril, por el que se aprueba el texto refundido de la Ley de Propiedad Intelectual, regularizando, aclarando y armonizando las disposiciones legales vigentes sobre la materia*. Última modificación: 31 de diciembre de 2011. [consulta: 13-09-2014]. Disponible en: https://www.boe.es/buscar/pdf/1996/BOE-A-1996-8930-consolidado.pdf.