Instituto Tecnológico de Costa Rica
Escuela de Ingeniería en Computación
Programa de Maestría

Implementación Eficiente en la Confección de un Frente de Pareto co-objetivo para el mecanismo de Matchtracking del Algoritmo *Fuzzy* ARTMAP

Tesis para optar por el grado de Magister Scientiae en Computación con énfasis en Ciencias de la Computación

Nuria Vanessa Figueroa Flores

Carnet 9833640

Cartago, Costa Rica

2014

Implementación Eficiente en la Confección de un Frente de Pareto co-objetivo para el mecanismo de Matchtracking del Algoritmo Fuzzy ARTMAP

Resumen

El mecanismo de *matchtracking* es parte integral del *Fuzzy* ARTMAP, itera sobre el conjunto de las plantillas buscando la plantilla $\mathbf{w}^*$ de la clase corregida que mejor representa al patrón de entrada $\mathbf{I}$. En este trabajo proponemos visualizar dicho mecanismo con una búsqueda dentro de un frente de pareto construido de forma apropiada para un problema de optimización co-objetiva univaluado.

Abstract

Matchtracking mechanism is integral to the Fuzzy ARTMAP, it iterates over the set of templates seeking the $\mathbf{w}^*$ corrected template class that best represents the input pattern $\mathbf{I}$. We propose to visualize this mechanism with a search within a Pareto front built appropriately to a problem in co-single valued objective optimization, which achieves an improvement in the complexity of $O(n)$ a $O(\log(n))$.

Palabras clave: Redes Neurales, ARTMAP, *Fuzzy* ARTMAP, *Matchtracking*, Frente de Pareto, Optimización Co-objetiva.

TEC | Tecnológico de Costa Rica

Programa de Maestría en Computación
Teléfono: 2550-2402

APROBACIÓN DE LA TESIS

"Implementación Eficiente en la Confección de un Frente de Pareto co-objetivo para el mecanismo de Matchtracking del Algoritmo Fuzzy ARTMAP"

TRIBUNAL EXAMINADOR

Ph.D. José Castro Mora
Profesor Asesor

Ph.D. José E. Araya Monge
Profesor Lector

Di. Juan Félix Ávila Herrera
Profesor Externo

M.Sc. Luis Carlos Loaiza Canet
Coordinador a.i. del Programa
de Maestría en Computación

Febrero, 2014

Dedicatoria

Este trabajo lo dedico a mis hijas Catalina y Sara, mis pequeñas gatitas cambiaron mi forma de ver el mundo.

Agradecimientos

Doy gracias infinitas a Dios por permitirme terminar este trabajo y a mi familia por el apoyo incondicional. Han sido muchos años e intentos por acabarlo que casi parece un sueño lograrlo.

Ephígrafe

“A veces sentimos que lo que hacemos es tan solo una gota en el mar, pero el mar sería menos si le faltara una gota”.

Madre Teresa de Calculta

Índice

1. Propuesta de Tesis **9**
1.1. Introducción 9
1.2. Objetivo general 15
1.2.1. Objetivos específicos 15

2. Marco Teórico **16**
2.1. Algoritmo *Fuzzy* ARTMAP 16
2.2. Arquitectura de la Red Neural *Fuzzy* ARTMAP 17
2.3. El mecanismo del *Matchtracking* 23

3. Frentes de Pareto y optimización co-objectiva **26**
3.1. *Matchtracking* y Frentes de Pareto 28
3.2. Implementación del Frente de Pareto 33

4. Resultados de desempeño **40**

5. Conclusiones y recomendaciones **47**

6. Bibliografía **49**

7. Anexos **53**
7.1. Algoritmo *Fuzzy* ARTMAP 53
7.2. Algoritmo FS-FAM modificado a partir del *Fuzzy* ARTMAP 66
7.3. Modificación propuesta Pareto FAM 71

1. Propuesta de Tesis

1.1. Introducción

En neurociencia, el aprendizaje se refiere a un proceso de cambio en un individuo que permite su adaptación al entorno. Estos cambios pueden ser neurales, cognitivos o conductuales pero son resultado de la experiencia. *"Learning "is making useful changes in the working of our minds*, con esta frase Marvin Minsky inicia su ensayo alrededor del significado del aprendizaje Minsky (1988), y es precisamente este concepto el que desencadena las discusiones, al trabajar en el contexto de la inteligencia artificial.

Al intentar emular el comportamiento del cerebro humano, se ha buscado sistematizar los variados fenómenos que conllevan al aprendizaje. Su taxonomía incluye, entre otros, el aprendizaje memorístico, el aprendizaje basado en ejemplos, el aprendizaje basado en reglas, el aprendizaje por refuerzo, pero es quizá el aprendizaje con redes neuronales el que ha buscado representar de forma más simple la estructura del cerebro.

En una red neuronal su unidad base, nodo, se activa tal y como actúa una neurona biológica al reaccionar ante un estímulo (entrada). Cuando esta activación sobrepasa cierto umbral transmite la señal al siguiente nodo de la red. A cada conexión de este tipo se le asigna un peso numérico que será modificado de manera tal que su respuesta se aproxime al esperado. Existen variedad de tipos de estructura de red, pero podemos distinguir entre aquellas cuya conexión es unidireccional (sin ciclos) y las recurrentes que con varias capas de nodos e interconexiones ofrecen una mayor variedad topológica.

Desde el punto de vista biológico, se suele aceptar que la información almacenada en el cerebro esta más relacionada con los valores sinápticos entre las neuronas que con ellas mismas. Por ello, cuando hablamos de redes neurales artificiales podemos decir que el co-

nocimiento se encuentra representado en el conjunto de pesos de las conexiones entre las neuronas de la red. Más aún, dado que todo proceso de aprendizaje implica cambios en estas conexiones, se puede afirmar que una red aprende en la medida que modifique los valores de los pesos de la red.

Otra de las características de la memoria humana es su capacidad para aprender nuevos conceptos sin necesidad de olvidar lo aprendido con anterioridad. Investigadores como Donald Hebb (1949), Frank Rosenblatt (1957), Marvin Minsky y Seymour Papert (1969) ofrecieron modelos que permitieran, en cierta medida, lograr esa habilidad. Incluso Paul Werbos (1974) propuso el algoritmo de retropropagación para esto; sin embargo, no fue comprendido hasta tiempo después. Y es en este dilema que nace en 1977 la Teoría de Resonancia Adaptativa.

Al desarrollar una red neural para desempeñar una tarea particular de clasificación, usualmente se toma un conjunto de ejemplos (patrones de entrenamiento) y luego se los utiliza para entrenar la red. Una vez que la red aprendió adecuadamente los ejemplos, los pesos de la red está fijados, y entonces el sistema puede ser utilizado para clasificar patrones no vistos previamente. Este escenario es aceptable cuando dentro de el dominio del problema es posible seleccionar un conjunto de patrones de entrenamiento, que una vez aprendidos, permitan a la red clasificar con bastante precisión patrones nuevos. Desafortunadamente, en muchas situaciones reales, el dominio del problema no evidencia este buen comportamiento.

Por ejemplo, suponga que una compañía desea entrenar una red que reconozca las siluetas de las partes requeridas para confeccionar alguno de los artículos de su línea de producción. Las imágenes de dichas piezas pueden usarse para entrenar a la red, tarea que consumirá tiempo dependiendo del tamaño de la red requerida. Después que la red

ha aprendido el conjunto de entrenamiento (con algún criterio de precisión), el periodo de entrenamiento finaliza y los pesos están fijados.

Ahora asuma que en cierto momento un nuevo producto es introducido, y la compañía desea agregar los componentes del nuevo producto al conocimiento almacenado en la red. Esto típicamente requerirá un reentrenamiento de la red usando los patrones previos y los nuevos, pues entrenar solo con los patrones nuevos puede resultar en aprender bastante bien los patrones nuevos, pero olvidar los viejos. Apesar que este reentrenamiento puede tomar aproximadamente el mismo tiempo que el entrenamiento original, es una tarea que consumiría tiempo valioso para la empresa. Cabe señalar que si a una red neural se le presenta un patrón no visto previamente, que sea ligeramente diferente de todos los patrones de entrenamiento, en la mayoría de los modelos de redes neurales no hay un mecanismo que reconozca lo novedoso de la entrada.

Lo que hemos descrito es lo que Grossberg llama el dilema de estabilidad-plasticidad Grossberg (1976). Este dilema puede ser replanteado formulando las siguientes preguntas:

- ¿Cómo puede un sistema de aprendizaje permanecer adaptativo (plástico) en respuesta a una entrada significativa, y aún mantenerse estable en respuesta a una entrada irrelevante?
- ¿Cómo sabe un sistema cuando debe cambiar de un modo estable a un modo plástico?
- ¿Cómo puede el sistema retener la información previamente aprendida, mientras continúa aprendiendo cosas nuevas?

En respuesta a estas preguntas Grossberg desarrolló la Teoría de Resonancia Adaptativa (ART).

ART obtiene su nombre por la forma particular en la cual el aprendizaje y las rellamadas interjuegan en estas redes. En física, la resonancia ocurre cuando una vibración de pequeña amplitud en la frecuencia apropiada causa una vibración de gran amplitud en un sistema eléctrico o mecánico. En una red ART, la información, en la forma de elementos procesados de salida, resuena hacia adelante y hacia atrás con los representantes o prototipos de las categorías que reconoce de la red. Si los patrones apropiados se concretan, una oscilación estable (la equivalencia de resonancia en redes neurales) sobreviene. Durante este periodo de resonancia, el aprendizaje o la adaptación puede ocurrir. Antes de que la red alcance un estado de resonancia, ningún aprendizaje tiene lugar porque el tiempo requerido para cambios en los pesos es mucho mayor que el que le toma a la red alcanzar la resonancia.

En las redes ART, un estado de resonancia puede lograrse de dos maneras. Si la red ha aprendido previamente a reconocer un patrón de entrada, entonces el estado de resonancia se logrará rápidamente cuando se le muestra la entrada. Durante la resonancia el proceso de adaptación reforzará la memorización del patrón almacenado. Si el patrón de entrada no se reconoce de inmediato, rápidamente la red buscará en los patrones que tiene almacenados su par. Si no logra coincidencia, la red entrará en resonancia, con lo cual el nuevo patrón es almacenado por primera vez. Así, la red responde rápidamente a los datos previamente almacenados, y aún se mantiene en capacidad de aprender datos nuevos.

ART propone emular el comportamiento de las neuronas humanas al incluir un mecanismo de retroalimentación entre las neuronas de su capa de salida y entrada, de tal forma que en su interacción se ajustan sus pesos hasta lograr un estado equilibrado. Así, cuando una nueva entrada es presentada, los nodos (categorías) de la red compiten y cooperan por representarla. Finalmente una categoría es la ganadora o bien nace una nueva, este mecanismo de competencia es denominado *matchtracking*.

ART fue introducido por Grossberg para describir como en una red neural el reconocimiento de categorías es organizado por la misma red. Desde entonces diversas arquitecturas de redes basadas en ART han sido propuestas, encontramos ejemplos en: Williamson (1996), Verzy, Heileman, Georgiopoulos, y Healy (1998), Carpenter y Ross (1995), Carpenter (1997), G. C. Anagnostopoulos y Georgiopoulos (2001). Algunas de estas arquitecturas se originaron de Carpenter Grossberg y sus colegas de Boston University, pero otros investigadores del campo han contribuido con su literatura al ART.

La primera red neural de arquitectura ART, llamada ART1, apareció en 1987 Carpenter y Grossberg (1987). Este modelo es una red neural no supervisada capaz de auto-organizar(por agrupamiento) colecciones arbitrarias de patrones de entrada binarios. Más tarde ese mismo año 1987, se introdujo ART2, cuya arquitectura es capaz de agrupar colecciones arbitrarias de patrones de entrada con valores reales ($\mathbb{R}$). La red ART se volvió obsoleta en 1991, cuando la más simple arquitectura del *Fuzzy* ART se dió a conocer Carpenter, Grossberg, y Rosen (1991). Al igual que el ART2, *Fuzzy* ART es capaz de agrupar patrones de entrada analógicos. Y para patrones binarios, la operación del Fuzzy ART se reduce al ART1.

Las arquitecturas ART1, ART2 y *Fuzzy* ART desarrollan un aprendizaje no supervisado, en ocasiones llamado auto-organizado. Bajo esta modalidad se utilizan patrones de entrenamiento no clasificado y no existe un procedimiento de enseñanza externo, esto es no hay un conocimiento previo. Una función de enseñanza determina como se deben adaptar los parámetros de la red basados en la naturaleza de los patrones de entrada. En este caso, el procedimiento de enseñanza resulta en la categorización interna de acuerdo a alguna medida de similitud entre los patrones. Esto es, patrones similares se agrupan juntos durante el entrenamiento de la red. Estos grupos (clusters) son entonces considerados como las

clases o categorías a las cuales los patrones de entrada serán asociados.

Por otro lado el aprendizaje supervisado requiere de un conjunto de patrones de entrenamiento de clasificación conocida y con un procedimiento de entrenamiento externo. Dicho procedimiento se usa para adaptar los pesos de la red de acuerdo a la respuesta que brinda a los patrones de entrenamiento. Normalmente, estos ajustes son proporcionales al error presentado mientras se intenta clasificar el actual patrón de entrada. El uso de aprendizaje supervisado puede separarse en dos fases: la de entrenamiento y la de ejecución. Para la fase de entrenamiento debe considerarse una muestra representativa del ambiente en el cual se espera que opere la red.

A partir del modelo ART se diseñó una arquitectura jerárquica llamada ARTMAP, que bajo condiciones de aprendizaje supervisado posee mecanismos de control para crear un conjunto de categorías de reconocimiento estables y de tamaño óptimo, al maximizar su predicción y minimizar el error asociado. Originalmente ARTMAP se utilizó con vectores (de entrada y salida) binarios, luego llegó *Fuzzy* ART para modelar el aprendizaje con entradas analógicas y binarias, modelo que posteriormente fue incorporado en la arquitectura *Fuzzy* ARTMAP. En el marco teórico expondremos el algoritmo *Fuzzy* ARTMAP y los detalles del mecanismo de *matchtracking*.

Por otro lado, en el mundo real normalmente los problemas de optimización toman en cuenta varios objetivos con el propósito de hallar una solución ideal. Siguiendo la idea de encontrar un óptimo tenemos el concepto de óptimo de Pareto. Este concepto de Economía se basa en la búsqueda de un equilibrio con mejor bienestar, y concluye con que la máxima prosperidad común se obtiene cuando ninguna persona puede aumentar su bienestar en un intercambio sin perjudicar a otra. Estas ideas se han generalizado y en la teoría de optimización multivariada, una solución es óptimo de Pareto si no hay otra solución que

mejore uno de los objetivos en el problema sin empeorar los otros. Con esto tenemos un conjunto de posibles soluciones, denominado frente de Pareto. En el capítulo 3 presentamos los principales resultados alrededor de óptimos de Pareto. Así que surge la pregunta, ¿es posible dentro del algoritmo *Fuzzy* ARTMAP visualizar el mecanismo de *matchtracking* de manera distinta? Luego de revisar la teoría de optimización co-objetiva proponemos la siguiente hipótesis:

El mecanismo de *matchtracking* del algoritmo *Fuzzy* ARTMAP puede ser sustituído por la búsqueda sobre un frente de pareto para un problema de optimización co-objetivo.

En el capítulo 4 se muestran los resultados del experimento seguido para probar su validez y las conclusiones en el capítulo 5. Finalmente en los anexos se incluye el código implementado.

1.2. Objetivo general

Establecer una modificación del mecanismo de *matchtracking* del algoritmo *Fuzzy* Artmap con el fin de implementarlo de una forma novedosa que presente mejoras teóricas a su desempeño.

1.2.1. Objetivos específicos

1. Estudiar y caracterizar el algoritmo *Fuzzy* Artmap, así como sus variaciones existentes.

2. Analizar el mecanismo de *matchtracking* del algoritmo *Fuzzy* Artmap con respecto a la optimización co-objetiva.

3. Proponer una variante del mecanismo de *matchtracking* del algoritmo *Fuzzy* Artmap, con el fin de establecer la viabilidad de una mejora a su tiempo de ejecución.

2. Marco Teórico

2.1. Algoritmo *Fuzzy* ARTMAP

Fuzzy ARTMAP propuesto por primera vez en Carpenter, Grossberg, Markuzon, Reynolds, y Rosen (1992) es un algoritmo de aprendizaje supervisado con redes neurales. Su arquitectura permite el aprendizaje incremental en línea, que además de ser un proceso en el que se garantiza la convergencia a una solución, también posee un detector de características que reconoce patrones significativamente diferentes de aquellos que se le han presentado con anterioridad. Incluso puede suministrar explicaciones para sus salidas, y en consecuencia resolver en parte el problema de opacidad de las redes neurales (esto es la inhabilidad de explicar las respuestas que produce). Algo por las cuales la mayoría de las redes neurales han sido criticadas (Carpenter y Tan (1995)). También, las redes neurales del *Fuzzy ARTMAP* tienen la la cualidad de incrementar dinámicamente su tamaño en tanto el proceso de aprendizaje progresa, y realizar este incremento solo cuando la tarea de aprendizaje lo requiere. Esta cualidad elimina la necesidad de especificar una arquitectura arbitraria de red neural previo al proceso de aprendizaje inicial.

A pesar que el *Fuzzy ARTMAP* ha sido utilizado mayoritariamente para clasificación (G. Anagnostopoulos (2000)), puede usarse como función de aproximación en ciertos problemas (Carpenter y Tan (1995), Azuaje (2001)).

En esta investigación estamos interesados en el comportamiento del mecanismo de *matchtracking* del *Fuzzy* ARTMAP, mecanismo utilizado en la mayoría de los descendientes del ARTMAP, de los cuales *Fuzzy* ARTMAP es uno. Concretamente utilizaremos el mecanismo de *matchtraking* del SFAM2.0 de Taghi, Baghmisheh, y Pavesic (2003) como base de nuestro análisis. De ahora en adelante, nos referiremos al SFAM2.0 como FS-FAM, y ocasionalmente como *Fuzzy* ARTMAP. Con base en lo planteado, los resultados

que presentamos aquí son válidos para cualquier red neural que utilice el mecanismo de *matchtracking*.

Cuando el *Fuzzy* ARTMAP está en el proceso de aprendizaje de un patrón de entrada **I**, las plantillas en su capa de categoría de representación compiten para representar al patrón nuevo, y eventualmente una de estas plantillas lo aprende. Esta competencia puede inducir el mecanismo de *matchtracking*: un mecanismo que itera sobre el conjunto de las plantillas buscando una plantilla $\mathbf{w}^*$ de la clase corregida que mejor representa al patrón de entrada **I**. En nuestra experiencia la iteración inducida no tiene un alto costo computacional, y por tanto, creemos que esta parece ser la razón por la cual ella no ha sido sujeto de un análisis riguroso de optimización.

2.2. Arquitectura de la Red Neural *Fuzzy* ARTMAP

La red *Fuzzy* ARTMAP posee una arquitectura auto-organizativa que es capaz de aprender a reconocer rápidamente un nuevo patrón de entrada. La red opera determinando automáticamente la generalización o precisión necesaria al momento de clasificar un patrón de entrada. En este proceso involucra operadores de lógica difusa, AND (*min*) y OR (*max*), que son usados para definir el rango de valores tolerados por una categoría. Una función de escogencia ayuda a determinar si una categoría en particular representa al patrón de entrada. El operador *min* ayuda a definir características que están presentes; mientras que, el operador *max* lo hace para con las características ausentes.

En 1993 (Kasuba (1993)) desarrolla una versión simplificada de la arquitectura original del *Fuzzy* ARTMAP , denominada FS-FAM. Esta propuesta reduce la sobrecarga computacional y aspectos redundantes del original pero sin perder la capacidad de reconocimiento de patrones. La figura 1 muestra un diagrama de bloques de los componentes del FS-FAM.

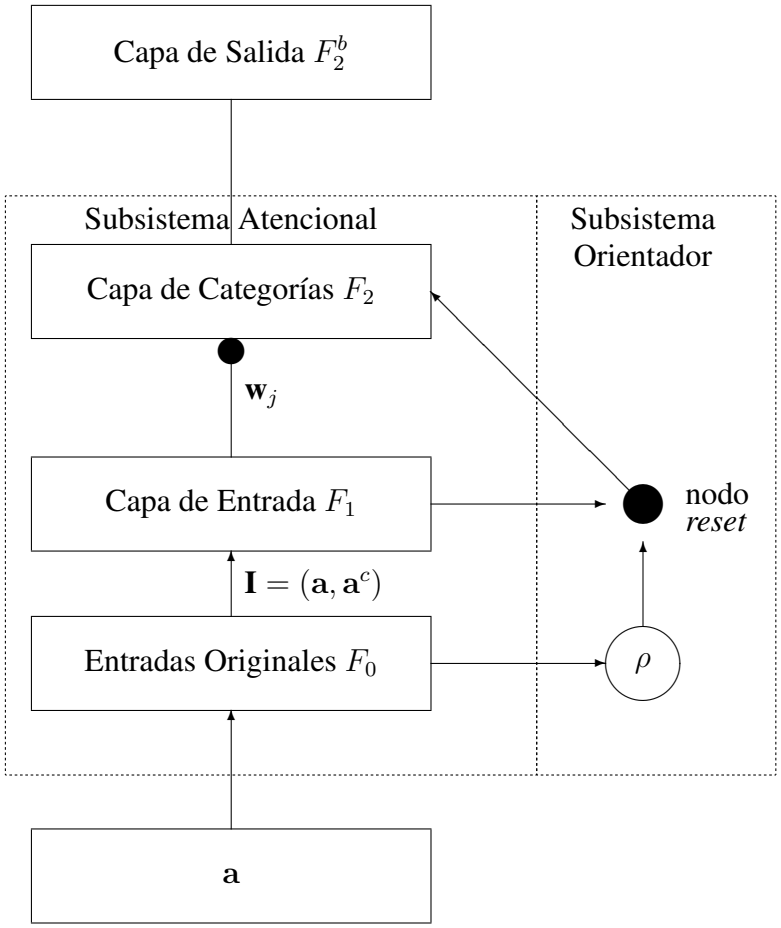


Figura 1: Diagrama de bloque del FS-FAM.

Su arquitectura posee tres capas de redes principales. La primer capa llamada *capa de entrada* (denotada por F_1), donde se presentan los patrones de entrada codificados (designados por $\mathbf{I}$). La segunda capa es la *capa de categorías de representación* (denotada por F_2), donde se crean las representaciones comprimidas (llamadas *categorías*) de estos patrones de entrada y designados por $\mathbf{w}_j$. Finalmente, la tercer capa es la *capa de salida* (denotada por F_2^b) y es la que mantiene las etiquetas de las categorías formadas en F_2.

El aprendizaje en FS-FAM ocurre modificando o ajustando los pesos asociados a cada $\mathbf{w}_j$, llamados *plantillas* de categorías o plantillas para abreviar. Todos los patrones de entrada $\mathbf{I}$ que son presentados en F_1 tienen la siguiente forma:

$$\mathbf{I} = (\mathbf{a}, \mathbf{a}^c) = (a_1, a_2, \ldots, a_M, a_1^c, a_2^c, \ldots, a_M^c)$$

donde,

$$a_i^c = 1 - a_i;\ \forall i \in \{1, 2, \ldots, M\}$$

Asumiremos que el vector de entrada $\mathbf{a}$ es tal que cada uno de sus componentes pertenece al intervalo [0, 1]. La construcción presentada arriba donde se crea $\mathbf{I}$ a partir de $\mathbf{a}$ es llamada *codificación complementaria* y se requiere para garantizar la operación exitosa del *Fuzzy* ARTMAP Carpenter y cols. (1991).

El FS-FAM puede ejecutarse en dos fases distintas: la *fase de entrenamiento* y la *fase productiva*. La fase de entrenamiento del FS-FAM puede describirse como sigue:

dado un conjunto de entradas *PT* y los pares de etiquetas asociadas a ellas, $\left\{(\mathbf{I}^1, \textit{etiqueta}(\mathbf{I}^1)), \ldots, (\mathbf{I}^{PT}, \textit{etiqueta}(\mathbf{I}^{PT}))\right\}$, se pretende entrenar al FS-FAM para asociar cada patrón de entrada del conjunto de entrenamiento con su correspondiente etiqueta.

Para ello se presenta repetidamente el conjunto de entrenamiento a la arquitectura del FS-FAM. Esto es, se muestra $\mathbf{I}^1$ a F_1, *etiqueta*($\mathbf{I}^1$) a F_2^b, $\mathbf{I}^2$ a F_1, *etiqueta*($\mathbf{I}^2$) a F_2^b, y finalmente $\mathbf{I}^{PT}$ a F_1, y *etiqueta* ($\mathbf{I}^{PT}$) a F_2^b.

Tal presentación del conjunto de entrenamiento al FS-FAM se realiza tantas veces como sea necesario, para que el FS-FAM clasifique correctamente todos los patrones de entrada. La meta se logra (esto es el aprendizaje es completo) cuando durante la presentación del conjunto de entrenamiento no se crean nuevos pesos y permanecen invariantes los demás pesos de la red.

Esta modalidad de entrenamiento es llamada *aprendizaje fuera de línea*. Existe otra modalidad que es llamada *entrenamiento en línea*, donde cada uno de los pares entrada/etiqueta son presentados al FS-FAM sólo una vez. Resumimos a continuación la fase de entrenamiento en línea del FS-FAM tal y como la presentan Taghi et al.en Taghi y cols. (2003).

1. Encontrar la categoría más cercana en la capa de representación de categorías del FS-FAM que “resuena ”con el patrón de entrada“.

2. Si la etiqueta de la categoría elegida como ganadora en el paso anterior y la etiqueta del patrón de la entrada coinciden, se actualiza la categoría escogida como la más cercana al patrón de entrada.

3. De otro modo, se descarta,y temporalmente se incrementa el umbral de resonancia (llamado *parámetro de vigilancia*), y se prueba con la siguiente categoría ganadora. Este proceso es llamado *matchtracking*.

4. Si ninguna categoría cumple con el *criterio de ajuste*, se creará una nueva categoría (asignando como representante de la nueva categoría al patrón de entrada, y desig-

nando como la etiqueta de la nueva categoría a la etiqueta del patrón de entrada).

La categoría más cercana al patrón de entrada $\mathbf{I}$ presentado al FS-FAM se determina encontrando la categoría que maximiza la función:

$$T(\mathbf{I}, \mathbf{w}_j, \alpha) = \frac{|\mathbf{I} \wedge \mathbf{w}_j|}{\alpha + |\mathbf{w}_j|} \tag{1}$$

Esta ecuación contempla dos operandos, uno de ellos es el *operador min difuso*, designado por el símbolo $\wedge$. El resultado de la operación min difusa de dos vectores $\mathbf{x}$, y $\mathbf{y}$, designada como $\mathbf{x} \wedge \mathbf{y}$, es un vector cuyos componentes son el mínimo de los correspondientes componentes de $\mathbf{x}$ y $\mathbf{y}$.

El otro operador está denotado por $|\cdot|$. Para un vector $\mathbf{x}$, $|\mathbf{x}|$ es llamado la norma del vector $\mathbf{x}$ y lo definimos como la suma de sus componentes.

La función $\mathbf{T}$ definida arriba es llamada función de activación o función de escogencia del patrón de entrada $\mathbf{I}$ con respecto al nodo j de F_2, donde su neurona de representación (*plantilla*) corresponde con el vector $\mathbf{w}_j$. Esta función obviamente depende del parámetro α de la red FS-FAM , llamado *párametro de escogencia*, que asume valores en el intervalo $(0, \infty)$. En la mayoría de las simulaciones del *Fuzzy* ARTMAP el rango utilizado para α es el intervalo $(0, 10]$ Georgiopoulos, Fernlund, Bebis, y Heileman (1996).

La resonancia de una categoría $\mathbf{w}_j$ para un patrón de entrada $\mathbf{I}$ se determina al examinar si la función $\rho(\mathbf{I}, \mathbf{w}_j)$, llamada *cociente de vigilancia*, y definida como

$$\rho(\mathbf{I}, \mathbf{w}_j) = \frac{|\mathbf{I} \wedge \mathbf{w}_j|}{|\mathbf{I}|} \tag{2}$$

satisface la siguiente condición:

$$\rho(\mathbf{I}, \mathbf{w}_j) \geq \rho \tag{3}$$

Si la inecuación se satisface decimos que la categoría $\mathbf{w}_j$ resuena con el patrón $\mathbf{I}$. El parámetro ρ es llamado *parámetro de vigilancia* y asume valores en el intervalo $[0, 1]$. Conforme el parámetro de vigilancia se incrementa, se crean más nodos de categorías en la capa de representación de categorías (F_2) del *Fuzzy* ARTMAP. Tenemos entonces que la función $T(\mathbf{I}, \mathbf{w}_j, \alpha)$ corresponde con el criterio de similitud por optimizar; mientras que de $\rho(\mathbf{I}, \mathbf{w}_j)$ se pretende una similitud mínima.

Al principio del entrenamiento con una sola entrada, la etiqueta del par de salida establece el valor de ρ igual al valor del parámetro de vigilancia base, $\bar{\rho}$, el cual es un valor escogido en el intervalo $[0, 1]$. Si la etiqueta del patrón de entrada ($\mathbf{I}$) es la misma que la de la categoría con la cual el patrón resuena, entonces la plantilla de la categoría ($\mathbf{w}_j$) es actualizada para incorporar las características de este nuevo patrón ($\mathbf{I}$).

La plantilla se actualiza de acuerdo con:

$$\mathbf{w}_j \leftarrow \mathbf{w}_j \wedge \mathbf{I} \tag{4}$$

El proceso de actualización de las plantillas, ilustrada en la ecuación anterior, ha sido llamado el *aprendizaje rápido* del *Fuzzy* ARTMAP. Nos concentramos solo con el aprendizaje rápido en el FS-FAM; sin embargo, vale la pena señalar que el aprendizaje lento, que se encuentra en Carpenter y cols. (1992), no compromete a los algoritmos propuestos en esta investigación.

Al principio del entrenamiento del FS-FAM se inicializan:

- el parámetro de escogencia α (escogido en el intervalo $(0, 10)$).
- el parámetro de vigilancia base $\bar{\rho}$ (escogido en $[0, 1]$).
- el nodo inicial $\mathbf{w}_0$ (escogido como el vector cuyas entradas son todas 1).

En el FS-FAM, $\mathbf{w}_0$ es la única plantilla con la cual se cuenta al principio del entrenamiento. Dado su valor (todas las entradadas son 1) se garantiza que pasará la prueba con el cociente de vigilancia.

Se creará una nueva plantilla $\mathbf{w}$ igual al patrón de entrada $\mathbf{I}$ siempre y cuando el nodo inicial $\mathbf{w}_0$ logre maximizar la función de activación *T*, esto es, que gane la competencia de activación. En el caso del FS-FAM, el nodo inicial $\mathbf{w}_0$ nunca es eliminado o actualizado, en lugar de ello es utilizado como asidero para cuando requiere crear nuevas plantillas.

2.3. El mecanismo del *Matchtracking*

En el proceso de aprendizaje, cuando la situación para un patrón de entrada $\mathbf{I}$ es que la plantilla $\mathbf{w}_j$ de la categoría escogida como la más cercana (ganadora de la competencia por representarlo), con $\mathbf{w}_j \neq \mathbf{w}_0$, y tal que la etiqueta de esta plantilla $\mathbf{w}_j$ es diferente de la etiqueta del patrón de entrada $\mathbf{I}$, entonces la plantilla es reinicializada y el parámetro de vigilancia ρ se incrementa al nivel:

$$\rho \leftarrow \rho(\mathbf{I}, \mathbf{w}_j) + \varepsilon \tag{5}$$

En la ecuación de arriba ε toma valores "muy pequeños ". Incrementando el valor del parámetro de vigilancia como se muestra en la ecuación 5 garantizamos que en la siguiente activación de la competición la última plantilla ganadora $\mathbf{w}_j$ se excluya. Es difícil establecer correctamente el valor de ε que garantice que después de reinicializar la categoría no pierda el FS-FAM una plantilla de categoría válida. Sin embargo, en la práctica, los valores típicos del parámetro ε se toman del intervalo $[0{,}00001, 0{,}001]$ Georgiopoulos, Dagher, Heilman, y Bebis (1999).

Luego de reinicializar la plantilla $\mathbf{w}_j$ (si es del caso), se buscan otras plantillas que maximicen la función de activación, mientras se satisface la condición con el parámetro de vigilancia.

Este proceso continua hasta que el nodo inicial $\mathbf{w}_0$ gana o se encuentra la plantilla de categoría que maximiza la función de activación, satisface la vigilancia y además tiene la misma etiqueta que el patrón de entrada presentado al FS-FAM. Una vez que esto ocurre, la correspondiente creación de una nueva plantilla o la actualización de una se sigue de la ecuación (4).

Este proceso iterativo de incrementar el parámetro ρ y buscar de nuevo por el ganador de la competencia es llamado *matchtracking* y es el segmento del algoritmo FS-FAM al cual se refiere nuestro análisis.

El seudocódigo para el algoritmo del FS-FAM (en su fase de entrenamiento en línea) se muestra en la Figura 2. En dicha fase el proceso de aprendizaje (de línea 3 a 13) revisa los datos en una sola ocasión. Presentamos la versión en línea porque es la más simple y la iteración sobre el conjunto de entrenamiento es irrelevante para el análisis del mecanismo de *matchtracking*. El mecanismo de *matchtracking* se presenta de la línea 5 a la 10.

El código para el algoritmo *Fuzzy* ARTMAP se muestra en el Anexo 1, corresponde al desarrollado por el investigador Rajeev Raizada sobre la propuesta original del CNS Technology LAB. El profesor Raizada ha colocado este material expresamente con fines académicos en el sitio de su laboratorio adscrito al Department of Brain and Cognitive Sciences at the University of Rochester (http://raizadalab.org/).Nuestra propuesta toma como base dicho algoritmo, modificándolo para que corresponda con el FS-FAM.

FAM–Learning $\left(\mathbf{I}^1, \ldots, \mathbf{I}^{PT}; \bar{\rho}, \alpha, \varepsilon\right)$
1. $\mathbf{w}_0 \leftarrow \underbrace{(1, 1, \ldots, 1)}_{2M}$
2. $templates \leftarrow \{\mathbf{w}_0\}$
3. for each $\mathbf{I} \in \left\{\mathbf{I}^1, \mathbf{I}^2, \ldots, \mathbf{I}^{PT}\right\}$
4. do $\rho \leftarrow \bar{\rho}$
5. repeat
6. $S \leftarrow \{\mathbf{w}_j : \rho(\mathbf{I}, \mathbf{w}_j) \geq \rho\}$
7. $j_{max} \leftarrow \mathbf{maxarg}_j T(\mathbf{I}, \mathbf{w}_j, \alpha) : \mathbf{w}_j \in S$
8. if *label*$(\mathbf{I}) \neq$ *label*$(\mathbf{w}_{j_{max}})$
9. then $\rho \leftarrow \rho(\mathbf{I}, \mathbf{w}_{j_{max}}) + \varepsilon$
10. until $(\mathbf{w}_{j_{max}} = \mathbf{w}_0)$ **or** $(label(\mathbf{I}) = label(\mathbf{w}_{j_{max}}))$
11. if $\mathbf{w}_{j_{max}} \neq \mathbf{w}_0$
12. then $\mathbf{w}_{j_{max}} \leftarrow \mathbf{w}_{j_{max}} \wedge I$
13. else $templates \leftarrow templates \cup \{I\}$
14. return $templates$

Figura 2: Algoritmo FS-FAM fase de entrenamiento en línea

3. Frentes de Pareto y optimización co-objectiva

Un problema de optimización multiobjetiva (MOP) de acuerdo con Coello, Lamont, y Veldhuizen (2001) toma en cuenta un vector de variables de decisión $\mathbf{x} = (x_1, \ldots, x_n)$ que optimiza al vector $\mathbf{y}$, compuesto por una serie de funciones objetivo f_i, como se muestran a continuación

$$\mathbf{f}(\mathbf{x}) = \mathbf{y} = (f_1(\mathbf{x}), \ldots, f_k(\mathbf{x}))$$

sujeto a un conjunto de desigualdades de restricción

$$g_i(\mathbf{x}) \geq 0, \ i \in \{1, \ldots, p\}$$

Las funciones f_i forman una descripción matemática del criterio con respecto a su desempeño, y usualmente tienen conflicto entre ellas. Por tanto *optimizar* al vector $\mathbf{f}(\mathbf{x})$ significa encontrar valores de las f_i que muestren un balance aceptable para quien toma la decisión final.

Dado que tenemos un número de funciones objetivo por optimizar, la noción de óptimo, como usualmente la entendemos, debe cambiar. El primero en reconocer este hecho fue Francis Ysidro Edgeworth en 1881, su planteamiento fue luego generalizado por Wilfrido Pareto, y por ello es ahora llamado *Edgeworth–Pareto optimum* o simplemente óptimo de Pareto.

Para introducir el concepto de frente de Pareto comenzaremos con una serie de definiciones. Pero antes es importante señalar que en optimización podemos buscar *disminuir* una función, por ejemplo de costo o *incrementar* una función, por ejemplo de utilidad. Tomaremos el punto de vista de incrementar una función (que en nuestro caso corresponde con la función de activación), dado que esta percepción es consistente con el del FS-FAM

al escoger la plantilla más cercana.

Definición 3.1. *Un vector* $\mathbf{u} = (u_1, \ldots, u_k)$ *se dice ser* (pareto) *dominante sobre otro vector* $\mathbf{v} = (v_1, \ldots, v_k)$ *si y sólo si* $\forall i \in \{1, \ldots k\}, u_i \geq v_i$, *y* $\exists j \in \{1, \ldots, k\} u_j > v_j$. *En este caso escribimos* $\mathbf{u} \succeq \mathbf{v}$.

Definición 3.2. *En un MOP decimos que un punto* $\mathbf{x} \in \mathcal{F}$ *es una* solución fuertemente dominante *si y sólo si no existe otro punto* $\mathbf{x}'$ *tal que* $f_i(\mathbf{x}) \leq f_i(\mathbf{x}')$ *para todo* i *y* $f_j(\mathbf{x}) < f_j(\mathbf{x}')$ *para algún* j.

Note que la definición de solución fuertemente dominante es muy similar a la de pareto dominante. La diferencia radica en que en pareto dominante comparamos los vectores, mientras que en soluciones fuertemente dominantes comparamos vectores de funciones objetivo. Esto nos guía a las definiciones formales de pareto óptimo, conjunto pareto óptimo, y frente de pareto.

Definición 3.3. *Un vector de variables de decisión* $\mathbf{x} = (x_1, \ldots, x_n)$ *es* pareto óptimo *si no existe otro vector* $\mathbf{x}' \in \mathcal{F}$, *donde* $\mathcal{F}$ *es la región factible, tal que* $f_i(\mathbf{x}) \leq f_i(\mathbf{x}')$ *para todo* i, *y* $f_j(\mathbf{x}) < f_j(\mathbf{x}')$ *para al menos un* j. *Como escribimos previamente, no existe* $\mathbf{x}'$ *tal que* $\mathbf{f}(\mathbf{x}') \succeq \mathbf{f}(\mathbf{x})$.

Esta definición básicamente establece que un vector es pareto óptimo si no es dominado por otro vector de la región factible. Tal propuesta usualmente no produce un solo valor óptimo, dando pie a la siguiente definición.

Definición 3.4. *Dado un MOP con vector objetivo* $\mathbf{y} = \mathbf{f}(\mathbf{x})$, *el* conjunto pareto óptimo $\mathcal{P}^*$ *se define como:*

$$\mathcal{P}^* = \{\mathbf{x} \in \mathcal{F} : \neg\exists \mathbf{x}' \in \mathcal{F} \cdot \ \mathbf{f}(\mathbf{x}') \succeq \mathbf{f}(\mathbf{x})\}$$

o simplemente el conjunto de los puntos pareto óptimo.

Definición 3.5. *Dado un MOP con $\mathcal{P}^*$ determinado, el frente de pareto $\mathcal{PF}^*$ se define como $\mathbf{f}(\mathcal{P}^*)$, esto es*

$$\mathcal{PF}^* = \{\mathbf{f}(\mathbf{x}) : \mathbf{x} \in \mathcal{P}^*\}$$

Un problema de optimización co-objetiva es un caso particular de la optimización múltiple, donde el vector $\mathbf{y} = \mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}))$. Bajo estas circunstancias podemos fácilmente graficar y visualizar el problema de optimización así como su frente de pareto.

Por ejemplo, en figura 3, los puntos del frente de pareto pueden identificarse por simple observación (esto son los puntos que no son dominados por otros puntos, como se estableció en las definiciones anteriores).

En la figura 3, los puntos del frente de pareto son distinguidos como puntos blancos, las líneas punteadas convergen a estos puntos del frente de pareto y ellas definen una región en el espacio que encierra todos los puntos para los cuales los puntos del frente son dominantes.

3.1. *Matchtracking* y Frentes de Pareto

De la explicación del algoritmo FS-FAM, y del seudo código que se muestra en la figura 2, vemos que el proceso de buscar la plantilla ganadora no es un problema de optimización co-objetiva, es más bien un proceso iterativo para hallar:

$$\max_{\mathbf{w}_j \in templates} \{T(\mathbf{I}, \mathbf{w}_j, \alpha)\}$$

sujeto a la restricción $\rho(\mathbf{I}, \mathbf{w}_j) \geq \rho$.

Cuando el máximo es encontrado, y sólo si dicho máximo no pertenece a la clase correcta el proceso iterará, incrementando el valor de ρ como se presentó en la ecuación 5 y

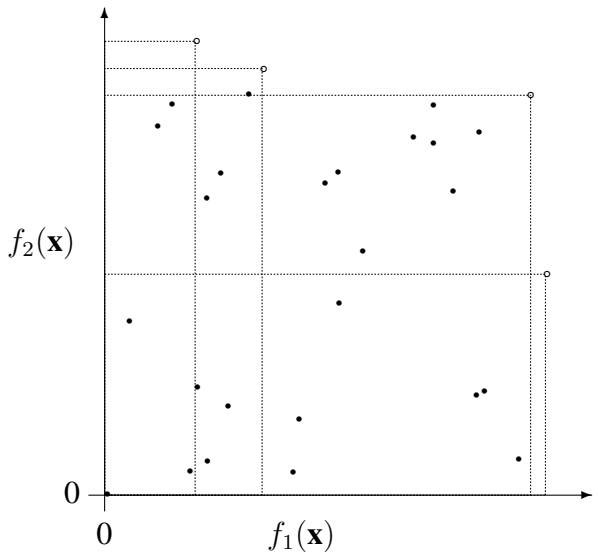


Figura 3: Gráfico de los puntos de la optimización co-objetiva presentando los puntos del frente de pareto como puntos blancos.

por tanto excluyendo la última plantilla ganadora de la competición.

De esta manera el FS-FAM es modelado como un problema de optimización univaluado sujeto a una desigualdad como restricción. Sin embargo, la iteración consecuencia del mecanismo de *matchtracking* cambia constantemente la región factible del problema. Y es en este contexto que el mecanismo de *matchtracking* se vuelve valioso para el modelo de optimización del FS-FAM visto como un problema de optimización co-objetiva con una región factible fija.

Considere el gráfico de la figura 4. Este gráfico muestra el valor de vigilancia contra las plantilla activadas en el problema del círculo dentro del cuadrado, también mencionado en Carpenter y cols. (1992). El problema del círculo dentro de un cuadrado es un problema *benchmark* "famoso "donde el clasificador busca discernir cuando un punto del espacio está dentro de un círculo (localizado dentro de un cuadrado) o fuera de él. Dicho círculo está centrado en el centro del cuadrado y su área es la mitad del área del cuadrado.

En esta figura, los puntos negros corresponden con plantillas de patrones localizados dentro del círculo y los puntos blancos son plantillas que representan patrones localizados fuera del círculo.

El gran punto negro ubicado en la parte alta de la línea punteada vertical representa una plantilla no seleccionada, cuya vigilancia es 1 pero su activación es menor a 0.5.

La línea horizontal punteada (al nivel $0{,}5$) representa la línea basal de vigilancia $\bar{\rho}$, que para este ejemplo se escogió igual a $0{,}5$. Las plantillas elegibles (estas son, plantillas que parecen codificar los patrones de entrada) están todas dentro del cuadrante noreste de la figura definido por las líneas $\rho = 0{,}5$ y $T \approx 0{,}5$.

Dentro de la región de plantillas elegibles (cuadrante noreste de la figura 4) la ganadora de la competencia será la plantilla con el mayor valor de activación $T(\mathbf{I}, \mathbf{w}_j, \alpha)$. Esta plantilla se muestra por las líneas punteadas que guían al punto.

Más aún, estas líneas definen la región de puntos (plantillas) que son pareto dominadas por dicha plantilla. Como esta plantilla ganadora no posee la clase correcta ocurriró *match-tracking*.

El proceso entonces incrementa el valor de ρ como se muestra en la ecuación 5, y si ε es lo suficientemente pequeño, todas las plantillas sobre la línea punteada horizontal serán plantillas elegibles durante la segunda revisión del conjunto de entrenamiento.

En tanto el proceso continúe eventualmente obtendremos una plantilla ganadora con la clase correcta, en caso contrario tendremos la plantilla no seleccionada (el gran punto negro

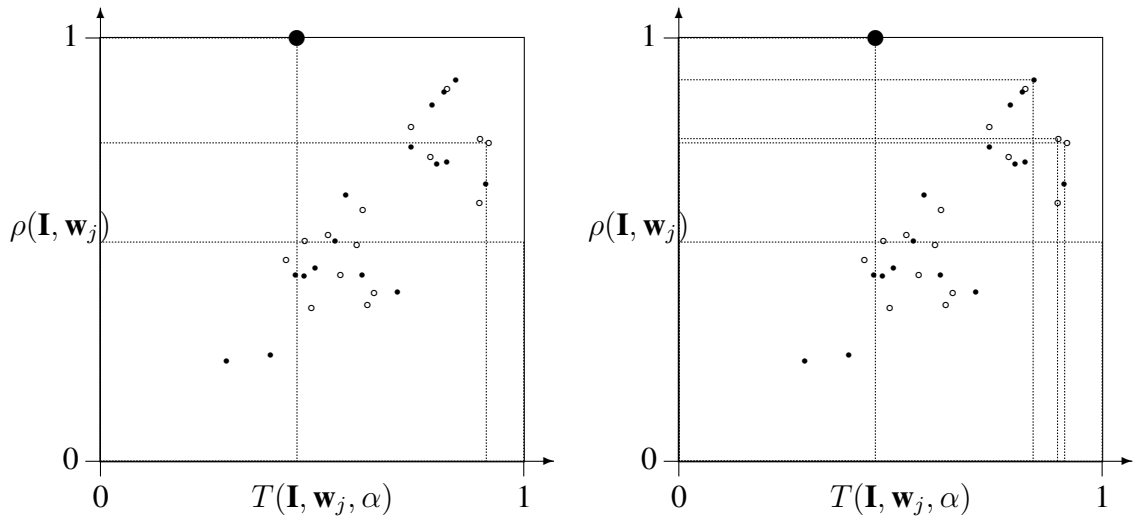


Figura 4: En este ejemplo, los puntos (blancos y negros) representan plantillas del problema del círculo dentro del cuadrado. El gran punto negro representa una plantilla no seleccionada. Las plantillas elegibles se localizan en el cuadrante definido por la línea de vigilancia igual a $0{,}5$ y la línea del valor de activación que es igual a $0{,}5$. En la figura de la izquierda, la plantilla señalada como un punto blanco con las líneas punteadas convergiendo a él, es la plantilla que primero fue escogida al presentar el patrón $\mathbf{I} = (0{,}13207, 0{,}661587)$, el cual reside dentro del círculo. La plantilla ganadora no es de la clase correcta (por cuanto esta plantilla representa patrones fuera del círculo) en consecuencia requerimos *matchtracking*. En la figura de la derecha se muestran los puntos de pareto óptimo (plantillas) que el FS-FAM examina (estos puntos tienen líneas punteadas que convergen a ellos). De estas plantillas, dos son de clases incorrectas (puntos blancos) y una es correcta (punto negro) que es la que eventualmente codifica el patrón de entrada dado.

de la parte superior de la figura) como la plantilla ganadora.

En este ejemplo la plantilla ganadora final posee la clase correcta, y se muestra en la figura 4 como un punto negro con líneas punteadas convergiendo a él.

El ejemplo presentado (en la figura 4) nos permite visualizar el resultado presentado formalmente en Castro, Figueroa, Georgiopoulos, y Secretan (2007) en el cual demostramos que los puntos elegidos por *matchtracking* forman parte de un frente de pareto de un problema de optimización co-objetiva particular.

La optimización en la elaboración de frentes de pareto se ha considerado ampliamente, y podemos citar Srinivasan y Rachmawati (2006), NPGA Horn, Nafpliotis, y Goldberg (1994), PESA-II Corne, Jerram, Knowles, y Oates (1994), PAES Knowles y Corne (1999), NSGA-II Deb, Pratap, Agarwal, y Meyarivan (2002), SPEA2 Zitzler, Laumanns, y Thiele (2002), HPMOEA Zou, Liu, Kang, y He (2004) y OMOEA-II Zeng, Yao, Kang, y Liu (2005) como algunos de los algoritmos que han mostrado mejor desempeño.

.

Teorema Toda plantilla $\mathbf{w}_{j_{max}}$ encontrada por el algoritmo de FS-FAM anterior es pareto óptima con respecto a las medidas de $\rho(\mathbf{I}, \mathbf{w})$ y $T(\mathbf{I}, \mathbf{w}, \alpha)$.

Demostración

Considérese el conjunto S que se define en la linea 6 del algoritmo. Claramente cualquier plantilla $\mathbf{w}_j$ o bien se encuentra en S, o se encuentra en su complemento S'. Por otra parte, demostrar que $\mathbf{w}_{j_{max}}$ es pareto óptima equivale a demostrar que dada una plantilla $\mathbf{w}_j$ cualquiera, ésta no domina a $\mathbf{w}_{j_{max}}$.

Sea entonces $\mathbf{w}_j$ una plantilla cualquiera tomada del conjunto de plantillas del FS-FAM, se consideran dos casos:

Caso 1: $\mathbf{w}_j \in S'$

como

$$S = \{\mathbf{w}_j : \rho(\mathbf{I}, \mathbf{w}_j) \geq \rho\}$$

Esto implica que

$$S' = \{\mathbf{w}_j : \rho(\mathbf{I}, \mathbf{w}_j) < \rho\}$$

y como $\mathbf{w}_{j_{max}} \in S$ entonces

$$\rho(\mathbf{I}, \mathbf{w}_{j_{max}}) \geq \rho > \rho(\mathbf{I}, \mathbf{w}_j) \Rightarrow \rho(\mathbf{I}, \mathbf{w}_{j_{max}}) > \rho(\mathbf{I}, \mathbf{w}_j)$$

Por lo tanto si $\mathbf{w}_j \in S'$ entonces $\mathbf{w}_j$ no domina a $\mathbf{w}_{j_{max}} \Rightarrow \mathbf{w}_{j_{max}}$ es pareto óptima.

Caso 2: $\mathbf{w}_j \in S$

Si $\mathbf{w}_j \in S$ entonces por la linea 7 del algoritmo sabemos que $T(\mathbf{I}, \mathbf{w}_{j_{max}}, \alpha) \geq T(\mathbf{I}, \mathbf{w}_j, \alpha)$ lo cual indica que $\mathbf{w}_{j_{max}}$ también es pareto óptima.

3.2. Implementación del Frente de Pareto

Para su implementación el algoritmo debe mantener registro de los puntos del frente de pareto que se encuentran en el cuadrante noreste de la figura 4. A pesar que puede verse como un alto costo computacional, dos factores hacen de este registro un proceso computacionalmente razonable.

Primeramente, está la correlación positiva entre los valores de activación y los valores

del cociente de vigilancia, que hace que las plantillas se distribuyan a lo largo de un vecindario alrededor de la línea del gráfico ganador (ver figura 4). Y por otro lado que el frente de pareto corta y cruza a la curva con una pendiente negativa. Esta combinación produce una cantidad pequeña de puntos en cualquier frente de pareto de un problema de *Fuzzy* ARTMAP comparado con el total de puntos elegibles del umbral definido por la línea base de vigilancia.

Es importante señalar que debemos mantener registro solo de las plantillas localizadas en el frente de pareto al que pertenece la categoría del patrón de entrada presentado, dado que estas plantillas son las únicas que eventualmente modificará el algoritmo.

Para todos las otras plantillas del frente de pareto necesitamos simplemente retener sus valores de activación y de vigilancia, lo cual será un vector bidimensional para cada plantilla de estas. Así la cantidad almacenada de plantillas de frentes de pareto no es excesiva y mantendremos un registro de todas las plantillas del FS-FAM.

Con el registro antes mencionado podemos seleccionar la plantilla correcta del FS-FAM, durante la presentación de un par (entrada/etiqueta de salida), en solo una lectura del conjunto de plantillas del FS-FAM.

Para algunos casos esto representa una ventaja en la implementación del FS-FAM. Por ejemplo, en un artículo relacionado Castro y cols. (2006) se propone una implementación en paralelo del algoritmo FS-FAM, para acelerar su fase de entrenamiento. Esta implementación por medio de *pipeline* permite un entrenamiento en línea paralelo eficiente al trabajar con grandes bases de datos. Sin embargo, dado que el paralelismo del mecanismo de *matchtracking* no es trivial, se utilizó un FS-MAP sin *matchtracking*.

```
Crear–Pareto {I, templates; α}
1. pareto ← {}
2. for each w ∈ templates
3. do p ← (T(I, w, α), ρ(I, w))
4.      Incluir ← TRUE
5.      for each x ∈ pareto
6.      do if p ≽ x
7.           then
8.                pareto ← pareto − {x}
9.           else
10.               if x ≽ p
11.                    then
12.                        Incluir ← FALSE
13.     if Incluir = True
14.          then pareto = pareto ∪ {p}
15. return pareto
```

Figura 5: Generación de un frente de pareto utilizando inserción en el frente en tiempo lineal $O(n)$

En la figura 5 presentamos seudocódigo para la creación de un frente de pareto. Este algoritmo efectúa, para cada patrón de entrada $\mathbf{I}$, una búsqueda lineal sobre los miembros del frente de pareto, lo cual claramente corresponde a una complejidad computacional de $O(n)$ (para cada patrón de entrada $\mathbf{I}$) donde n representa el número de elementos en el frente.

Sin embargo, esta complejidad se puede reducir analizando la estructura del frente y tomando ventaja de sus cualidades, cuando se trata específicamente de un problema de optimización co–objetiva.

A modo de ejemplo, consideremos el frente de pareto para un problema de *Fuzzy* ART-MAP presentado en la figura 6. Es un frente con cuatro puntos, que siguiendo el sentido de las manecillas del reloj, denotamos por A, B, C y D.

Ahora bien, dado un punto arbitrario $\mathbf{x}$ que queremos determinar si pertenece o no al

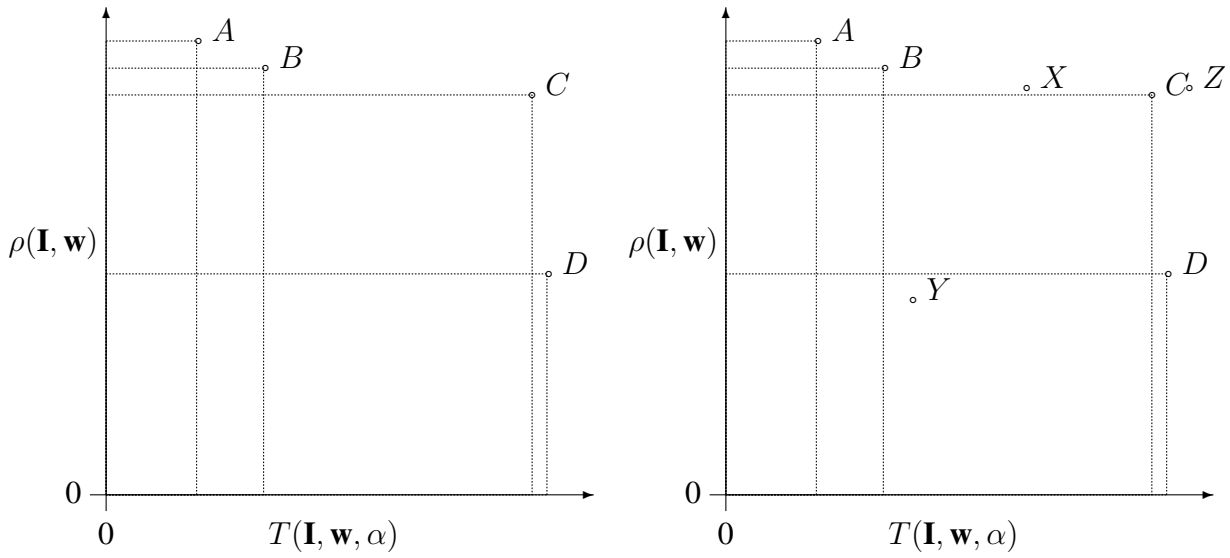


Figura 6: A la IZQUIERDA se encuentra un gráfico de los puntos de una optimización coobjetiva presentando los puntos del frente de pareto como puntos blancos. A la DERECHA se encuentra el mismo ejemplo, presentando tres puntos X, Y, Z que representan los tres casos distinto que se pueden dar a la hora de intertar en un frente de pareto.

frente y de este modo incluirlo o no, podemos dividir su análisis en tres casos:

- A) El punto $\mathbf{x}$ forma parte del frente y no domina a ningún otro punto del frente.
- B) El punto $\mathbf{x}$ forma parte del frente y domina por lo menos a un punto del frente
- C) El punto $\mathbf{x}$ no forma parte del frente.

Esto significa que $\mathbf{x}$ está dominado por al menos un punto del frente.

Para analizar cada caso consideremos hipotéticamente tres puntos denotados por X, Y, Z y ubicados según la figura 6 en un frente de pareto que actualmente se conforma por cuatro puntos denominados A, B, C y D. Analizaremos ahora su repectiva pertenencia a dicho frente.

Ahora bien, si revisamos la figura 6 y ordenamos los puntos del frente de pareto ascendentemente con respecto a su valor T, podemos ver que para los cuatro puntos del frente la

secuencia de orden según T es (A, B, C, D).

Correspondientemente si los ordenamos ascendentemente con respecto a ρ la secuencia es (D, C, B, A). Note que esta última está en orden inverso con respecto a la de T, esto ocurre siempre y cuando trabajemos con una función co-objetiva.

Por lo anterior preferimos considerar para ρ la secuencia invertida (A, B, C, D) y quedamos con una única represencación para ambos ordenamientos, uno ascendente, y el otro descendente.

Como la secuencia se encuentra ordenada en ambas coordenadas (una ascendente y la otra descendentemente), dado un punto arbitrario $\mathbf{x} = (T(\mathbf{I}, \mathbf{w}, \alpha), \rho(\mathbf{I}, \mathbf{w}))$ podemos buscar su ubicación dentro de la secuencia con respecto a la primer coordenada (valor T) o a su segunda coordenada (valor ρ). Denotaremos estos índices como i en el caso de la ubicación de $\mathbf{x}$ dentro de la secuencia según T y como j para con ρ.

Así, por ejemplo, en el caso del punto A tenemos $i = 1$ y $j = 1$. Ahora bien, consideraremos los casos enumerados anteriormente utilizando los puntos de ejemplo en la figura 6 para ilustrar el algoritmo.

Caso 1. El punto forma parte del frente y podemos ver que no domina a ningún otro punto. Este es el caso del punto X. Para dicho punto el vector de orden con respecto a T sería (A, B, X, C, D), mientras que con respecto a ρ tenemos (A, B, X, C, D). Los índices serían $i = 3$ y $j = 3$, los cuales coinciden. Entonces vamos a insertar al punto X en el frente y no eliminaremos punto alguno del frente original.

Caso 2: El punto forma parte del frente y es dominante sobre otros puntos del frente.

Crear–Pareto $(\mathbf{I}, \textit{templates}; \alpha)$
1. *pareto* $\leftarrow$ $\langle\rangle$ // `secuencia nula`
2. for each $\mathbf{w} \in$ *templates*
3. do $\mathbf{p} \leftarrow (T(\mathbf{I}, \mathbf{w}, \alpha), \rho(\mathbf{I}, \mathbf{w}))$
4. $\quad i \leftarrow$ INDEXT $(\mathbf{p}, \textit{pareto})$
5. $\quad j \leftarrow$ INDEXRHO $(\mathbf{p}, \textit{pareto})$
6. $\quad$ if $j \leq i$
7. $\quad\quad$ then
8. $\quad\quad\quad$ DELETE $(\textit{pareto}, j, i)$
9. $\quad\quad\quad$ INSERT $(\textit{pareto}, j, \mathbf{p})$
10. return *pareto*

Figura 7: Generación de un frente de pareto utilizando inserción en el frente en tiempo $O(\log(n))$

Este caso corresponde al punto Z de la figura 6, el cual domina los puntos C y D. Los vectores de orden según T y ρ serían (A, B, C, D, Z) y (A, B, Z, C, D) respectivamente, lo cual indica que $i = 5$ y $j = 3$. En este caso lo que corresponde es incluir el punto Z en el frente y eliminar los puntos C y D ya que se encuentran dominados por Z. Ahora bien, C y D tienen en la secuencia de orden según T y ρ las posiciones 3 y 4, valores que se encuentran entre los valores de j e i. Más específicamente, eliminamos los puntos K del frente, con índices k donde k cumple con la condición $3 = j \leq k < i = 5$.

Caso 3: El punto no forma parte del frente y es dominado por algún(os) puntos de él. Este es el caso del punto Y de la figura 6 el cual es dominado por los puntos C y D. En este caso los vectores de orden según T y ρ serían (A, B, Y, C, D) y (A, B, C, Y, D), los índices serían $i = 3$ y $j = 4$. No vamos a insertar el punto en el frente, ni a eliminar alguno de los que originalmente pertenecen a él.

Luego de analizar estos casos vemos que la clave está en comparar los índices que obtiene el punto $\mathbf{x}$ para el cual se discute su pertenencia al frente de pareto del problema.

Definimos como i al indice de la posición del punto $\mathbf{x}$ en el vector de orden según

$T(\mathbf{I}, w, \alpha)$ y j como al indice de la posición del punto $\mathbf{x}$ en el vector de orden según $\rho(\mathbf{I}, w)$ invertido, si $i < j$ entonces el punto no forma parte del frente, pero si $i \geq j$ entonces se inserta el punto $\mathbf{x}$ en el frente de pareto en la posición j y además se eliminan del frente todos los puntos que tienen indice k en el vector de orden según T (con $j \leq k < i$).

Ahora bien, como los índices se encuentran ordenados, la búsqueda en ellos se puede efectuar en tiempo $\log(n)$ donde n es el tamaño del frente de pareto. Esta propiedad permite proponer el algoritmo de la figura 7 para la inserción de los puntos en el frente.

En este algoritmo la estructura de datos *pareto* es una secuencia con índices numéricos que contiene los valores T y ρ de los puntos pertenecientes al frente.

La función INDEXT devuelve la posición i dentro de la secuencia *pareto* en la cual se debe insertar el valor $\mathbf{p}$ utilizando su valor $T(\mathbf{I}, \mathbf{w}, \alpha)$ de referencia y asumiendo que la secuencia *pareto* se encuentra ordenada *ascendentemente* con respecto a este valor.

La función INDEXRHO devuelve la posición j dentro de la secuencia *pareto* en la cual se debe insertar el punto $\mathbf{p}$ utilizando su valor $\rho(\mathbf{I}, \mathbf{w})$ como referencia y asumiendo que la secuencia *pareto* se encuentra ordenada *descendentemente* con respecto a este valor.

La función DELETE elimina todos los puntos en la secuencia mayores o iguales a j pero menores estrictos que i; y por último, INSERT inserta en la secuencia *pareto* un punto $\mathbf{p}$ en la posición j.
El código para la implementación del frente se encuentra en el Anexo 2 y fue implementado usando MATLAB 6.5.

4. Resultados de desempeño

Para este apartado hemos considerado tres bases de prueba para el problema del círculo dentro del cuadrado. En cada una los puntos se generaron de manera aleatoria y para cada uno se calculó si estaba dentro o no del círculo. La primera, D_{100}, contiene 100 entradas; mientras que las otras dos D_{1000} y D_{10000} agrupan 1000 y 10000 entradas respectivamente. Utilizamos una computadora portátil AMD Turion (tm) IIP540 Dual-core de 2.40GHZ, 4.0GB de RAM y con Windows 7.

Cargando en el espacio de trabajo del MATLAB la base correspondiente, ejecutamos primero el algoritmo FS-FAM y almacenamos su vector de pesos de categorías. Luego de limpiar el espacio de trabajo, volvemos a cargar la base y ejecutamos al algoritmo Pareto FAM, igual que antes almacenamos el vector de pesos de categorías. Para comparar, procedimos con la resta de los dos vectores almacenados.

El proceso anterior se siguió también con D_{1000} y D_{10000}. En cada caso el algoritmo Pareto FAM arrojó los mismos resultados que el algoritmo FS-FAM, la diferencia de los vectores de salida siempre fue un vector nulo. A modo de ejemplo mostramos a continuación las salida que se logra con en la última iteración para la base D_{100} con cada algoritmo.

```
Salida para FS-FAM
>> iteration:
   100

  Columns 1 through 12

    0.0005    0.0917    0.1048    0.1052    0.1670    0.1720    0.2291    0.2334    0.2661    0.2782    0.2981    0.3141
    0.7880    0.8436    0.0145    0.6013    0.8081    0.0824    0.6861    0.3339    0.4635    0.3484    0.1523    0.3227
    0.9995    0.9083    0.0129    0.0631    0.8330    0.8280    0.7709    0.7666    0.7339    0.7218    0.7019    0.6859
    0.2120    0.1564    0.4374    0.0119    0.1919    0.9176    0.3139    0.6661    0.5365    0.6516    0.8477    0.6773
    1.0000    1.0000         0         0    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000

  Columns 13 through 24

    0.3213    0.3263    0.3414    0.3618    0.3635    0.3789    0.3979    0.4021    0.4175    0.4305    0.4324    0.4333
    0.9257    0.0427    0.1364    0.0134    0.8297    0.4398    0.3951    0.0508    0.4908    0.8407    0.0195    0.8781
    0.6787    0.6737    0.6586    0.6382    0.6365    0.6211    0.6021    0.5979    0.5825    0.5695    0.5676    0.5667
    0.0743    0.9573    0.8636    0.9866    0.1703    0.5602    0.6049    0.9492    0.5092    0.1593    0.9805    0.1219
    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000

  Columns 25 through 36
```

```
    0.4349    0.4779    0.5119    0.5288    0.5418    0.5806    0.6251    0.6416    0.6896    0.7240    0.7368    0.7488
    0.0308    0.8829    0.0074    0.8507    0.2968    0.7870    0.1322    0.7147    0.2209    0.6315    0.2464    0.0673
    0.5651    0.5221    0.4881    0.4712    0.4582    0.4194    0.3749    0.3584    0.3104    0.2760    0.2632    0.2512
    0.9692    0.1171    0.9926    0.1493    0.7032    0.2130    0.8678    0.2853    0.7791    0.3685    0.7536    0.9327
    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000

  Columns 37 through 48

    0.7493    0.7588    0.7657    0.7714    0.8187    0.8628    0.8641    0.8653    0.8984    0.9189    0.9307    0.9510
    0.0661    0.0219    0.0669    0.1380    0.4981    0.4784    0.4675    0.9775    0.9347    0.1489    0.1471    0.2447
    0.2507    0.2412    0.2343    0.2286    0.1813    0.1372    0.1359    0.1347    0.1016    0.0811    0.0693    0.0490
    0.9339    0.9781    0.9331    0.8620    0.5019    0.5216    0.5325    0.0225    0.0653    0.8511    0.8529    0.7553
    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000

  Columns 49 through 53

    0.9644    0.9731    0.9826    0.9851    1.0000
    0.9188    0.4805    0.1806    0.7118    1.0000
    0.0356    0.0269    0.0174    0.0149    1.0000
    0.0812    0.5195    0.8194    0.2882    1.0000
    1.0000    1.0000    1.0000    1.0000   -1.0000

FS-FAM finished.

elapsed_time =

    0.0150

Salida Pareto FAM
iteration:
   100

  Columns 1 through 12

    0.0005    0.0917    0.1048    0.1052    0.1670    0.1720    0.2291    0.2334    0.2661    0.2782    0.2981    0.3141
    0.7880    0.8436    0.0145    0.6013    0.8081    0.0824    0.6861    0.3339    0.4635    0.3484    0.1523    0.3227
    0.9995    0.9083    0.0129    0.0631    0.8330    0.8280    0.7709    0.7666    0.7339    0.7218    0.7019    0.6859
    0.2120    0.1564    0.4374    0.0119    0.1919    0.9176    0.3139    0.6661    0.5365    0.6516    0.8477    0.6773
    1.0000    1.0000         0         0    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000

  Columns 13 through 24

    0.3213    0.3263    0.3414    0.3618    0.3635    0.3789    0.3979    0.4021    0.4175    0.4305    0.4324    0.4333
    0.9257    0.0427    0.1364    0.0134    0.8297    0.4398    0.3951    0.0508    0.4908    0.8407    0.0195    0.8781
    0.6787    0.6737    0.6586    0.6382    0.6365    0.6211    0.6021    0.5979    0.5825    0.5695    0.5676    0.5667
    0.0743    0.9573    0.8636    0.9866    0.1703    0.5602    0.6049    0.9492    0.5092    0.1593    0.9805    0.1219
    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000

  Columns 25 through 36

    0.4349    0.4779    0.5119    0.5288    0.5418    0.5806    0.6251    0.6416    0.6896    0.7240    0.7368    0.7488
    0.0308    0.8829    0.0074    0.8507    0.2968    0.7870    0.1322    0.7147    0.2209    0.6315    0.2464    0.0673
    0.5651    0.5221    0.4881    0.4712    0.4582    0.4194    0.3749    0.3584    0.3104    0.2760    0.2632    0.2512
    0.9692    0.1171    0.9926    0.1493    0.7032    0.2130    0.8678    0.2853    0.7791    0.3685    0.7536    0.9327
    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000
```

```
  Columns 37 through 48

    0.7493    0.7588    0.7657    0.7714    0.8187    0.8628    0.8641    0.8653    0.8984    0.9189    0.9307    0.9510
    0.0661    0.0219    0.0669    0.1380    0.4981    0.4784    0.4675    0.9775    0.9347    0.1489    0.1471    0.2447
    0.2507    0.2412    0.2343    0.2286    0.1813    0.1372    0.1359    0.1347    0.1016    0.0811    0.0693    0.0490
    0.9339    0.9781    0.9331    0.8620    0.5019    0.5216    0.5325    0.0225    0.0653    0.8511    0.8529    0.7553
    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000

  Columns 49 through 53

    0.9644    0.9731    0.9826    0.9851    1.0000
    0.9188    0.4805    0.1806    0.7118    1.0000
    0.0356    0.0269    0.0174    0.0149    1.0000
    0.0812    0.5195    0.8194    0.2882    1.0000
    1.0000    1.0000    1.0000    1.0000   -1.0000

Pareto FAM finished

elapsed_time =

    0.0460
```

Tenemos así que la propuesta con el frente de Pareto ofrece los mismos resultados que el algoritmo FS-FAM. A continuación presentamos tres tablas del tiempo de ejecución, dado en segundos, tras diez presentaciones a cada algoritmo de la colección de puntos indicada.

Presentación de la Base D_{100}

Base D_{100}	FS-FAM	Pareto FAM
1	0.0160	0.030
2	0.0190	0.0340
3.	0.0190	0.0340
4	0.0200	0.0370
5	0.0160	0.0340
6	0.0140	0.040
7	0.0200	0.0280
8	0.0300	0.0330
9	0.0200	0.0370
10	0.0160	0.0350

Presentación de la Base D_{1000}

Base D_{1000}	FS-FAM	Pareto FAM
1	0.2920	0.9290
2	0.3090	0.9230
3	0.2940	0.9360
4	0.3030	0.9230
5	0.3110	0.9450
6	0.2900	0.9360
7	0.2910	0.9140
8	0.2920	0.9210
9	0.3030	0.9240
10	0.3110	0.9160

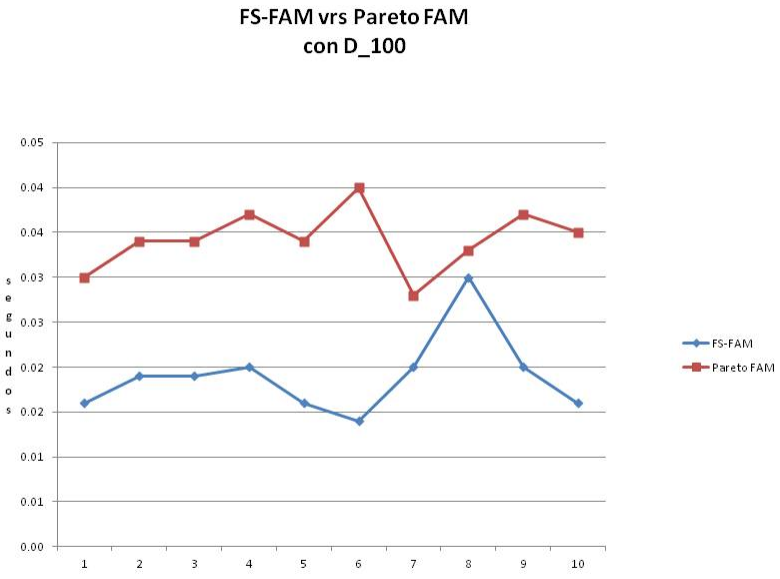
Presentación de la Base D_{10000}

Base D_{10000}	FS-FAM	Pareto FAM
1	19.4210	73.1050
2	19.2980	72.9240
3	19.2150	72.9460
4	19.2870	72.8840
5	19.2330	73.1540
6	19.2830	72.8540
7	19.2720	73.1330
8	19.2670	72.8280
9	19.2620	73.0460
10	19.4000	72.9470

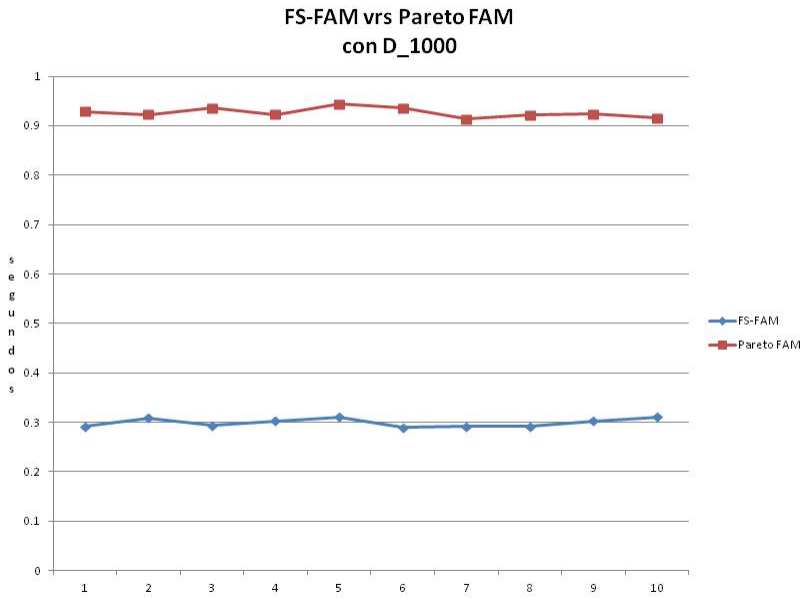
En el caso del Pareto FAM, los resultados anteriores evidencian mayor tiempo de ejecución que con FS-FAM; sin embargo, su estabilidad es comparable con él.

Gráficamente tenemos:

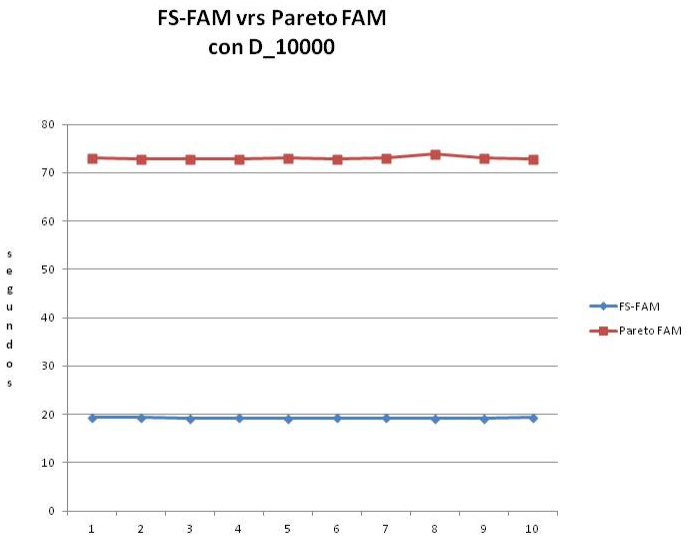
Desempeño del Pareto FAM versus FS-FAM con la base D_{100}



Desempeño del Pareto FAM versus FS-FAM con la base D_{1000}



Desempeño del Pareto FAM versus FS-FAM con la base D_{1000}



De los tiempos se observa que el algoritmo Pareto FAM demanda casi que el triple de tiempo pero con una estabilidad comparable.

5. Conclusiones y recomendaciones

El aporte de esta tesis ha sido que logramos hacer la implementación del frente de pareto para el algoritmo FSFAM. Demostramos también que nuestra propuesta obtiene los mismos resultados que el FSFAM. Además caracterizamos teóricamente el FSFAM con la teoría co-objetivo, aspecto que no se había hecho antes.

Los resultados experimentales indican que las implementaciones logran exactamente los mismos resultados que el algoritmo original; y apesar que no se obtiene una mejora en tiempo de ejecución consideramos que su desempeño puede mejorarse al experimentar más con el tipo de problema e implementación en una versión más reciente de MATLAB o otros lenguajes como C, con la intención de mejorar este resultado y distinguir en cuales situaciones nuestra variante tiene ventaja sobre el algoritmo original.

Por otro lado, al estudiar variaciones del *Fuzzy* ARTMAP, como el *Gaussian* ARTMAP Williamson (1996), vemos que es una síntesis de un clasificador gaussiano y una red neural de arquitectura ART. En él se propone la función de escogencia como una función discriminante de un clasificador gaussiano con distribuciones separadas, pero con la misma función de asociación. Con lo cual nuestra propuesta es viable de extender a esta variante del *Fuzzy* ARTMAP. Las ecuaciones de aprendizaje del Gaussian ARTMAP son sensiblemente más complejas que las del Fuzzy ARTMAP pero su desempeño ha mostrado ser mejor en bases cargadas de ruido Williamson (1997). La modificación de este algoritmo queda abierta para otro trabajo de investigación.

Queda por citar al *Ellipsoid* ARTMAP, algoritmo que se propone en G. C. Anagnostopoulos y Georgiopoulos (2001) junto con Ellipsoid ART, sus raíces son el fuzzy ARTMAP pero con la representación geométrica de hiperelipsoides en lugar de hiperectángulos. Sus autores los refieren como generalizaciones del *Hipersphere* ART e *Hipersphere* ARTMAP G. C. Anagnostopoulos y Georgiopoulos (2000). En ellos, durante la fase de entrenamiento, se establecen dos condiciones: i) los hiperelipsoides deben mantener constante el cociente

entre su eje mayor y los demás ejes, ii) debe mantener constante la dirección de su eje mayor una vez que es éste es establecido. A pesar de las anteriores limitaciones las categorías pueden tener orientaciones arbitrarias con la intención de capturar la mayor cantidad de características de los datos. En este caso la modificación Pareto FAM puede ser igualmente implementada, y dejamos tal tarea abierta a un estudio futuro.

6. Bibliografía

Referencias

Anagnostopoulos, G. (2000). *Novel approaches in adaptive resonance theory for machine learning*. Tesis Doctoral no publicada, Computer Engineering, UCF.

Anagnostopoulos, G. C., y Georgiopoulos, M. (2000). Hypersphere ART and ARTMAP for unsupervised and supervised incremental learning. En *Proceedings of the IEEE–INNS–ENNS* (Vol. 6, pp. 59–64). Como, Italy: IEEE–INNS–ENNS.

Anagnostopoulos, G. C., y Georgiopoulos, M. (2001). Ellipsoid ART and ARTMAP for incremental unsupervised and supervised learning. En *Proceedings of the IEEE–INNS–ENNS* (Vol. 2, pp. 1221–1226). Washington DC: IEEE–INNS–ENNS.

Azuaje, F. (2001, 3). A computational neural approach to support the discovery of gene function and classes of cancer. *IEEE Transactions on Biomedical Engineering*, *48*(3), 332–339.

Carpenter, G. A. (1997). Distributed learning, recognition and prediction by art and artmap neural networks. *Neural Networks*, *10*(8), 1473–1494.

Carpenter, G. A., y Grossberg, S. (1987). A massively parallel achitecture for a self-organizing neural pattern recognition machine. *Computer Vision, Graphics and Image Processing*, *37*, 54-115.

Carpenter, G. A., Grossberg, S., Markuzon, N., Reynolds, J. H., y Rosen, D. B. (1992, September). Fuzzy ARTMAP: A neural network architecture for incremental supervised learning of analog multidimensional maps. *IEEE Transactions on Neural Networks*, *3*(5), 698–713.

Carpenter, G. A., Grossberg, S., y Rosen, D. B. (1991). Fuzzy ART: Fast stable learning and categorization of analog patterns by an adaptive resonance system. *Neural Networks*, *4*(6), 759–771.

Carpenter, G. A., y Ross, W. D. (1995). ART–EMAP: A neural network architecture for ob-

ject recognition by evidence accumulation. *IEEE Transactions on Neural Networks*, *6*(5), 805–818.

Carpenter, G. A., y Tan, A. H. (1995). Rule extraction: From neural network architecture to symbolic representation. *Connection Science*, *7*(1), 3–27.

Castro, J., Figueroa, N., Georgiopoulos, M., y Secretan, J. (2007, 10). Implementación eficiente en la confección de un frente de pareto co-objetivo para el mecanismo de matchtracking del algoritmo fuzzy artmap. En *San josé xxxiii conferencia latinoamericana de informática.*

Castro, J., Georgiopoulos, M., Secretan, J., DeMara, R., Anagnostopoulos, G., y Gonzalez, A. (2006, January). Pipelining of fuzzy artmap without matchtracking: Correctness, performance bound, and beowulf evaluation. *Neural Networks Journal*, *20*(1), 109–128. Descargado de `http://dx.doi.org/10.1016/j.neunet.2006.10.003`

Coello, C. A., Lamont, G. B., y Veldhuizen, D. A. V. (2001). *Evolutionary algorithms for solving multi.objective problems*. Springer.

Corne, D. W., Jerram, N., Knowles, J., y Oates, M. (1994). PESA-II: Region-based selection in evolutionary multiobjective optimization. En *Proceedings of the genetic and evolutionary computation conference (gecco-2001)* (p. 283-290). Morgan Kaufmann.

Deb, K., Pratap, A., Agarwal, S., y Meyarivan, T. (2002). A fast and elititst multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, *6*(2), 182-197.

Georgiopoulos, M., Dagher, I., Heilman, G., y Bebis, G. (1999). *Properties of learning of a fuzzy art variant.* Descargado de `citeseer.nj.nec.com/georgiopoulos99properties.html`

Georgiopoulos, M., Fernlund, H., Bebis, G., y Heileman, G. L. (1996). Order of search in fuzzy art and fuzzy artmap: Effect of the choice parameter. *Neural Networks*, *9*(9),

1541–1559.

Grossberg, S. (1976). Adaptive pattern recognition and universal encoding ii: Feedback, expectation, olfaction, and illusions. *Biological Cybernetics*, *23*, 187–202.

Horn, J., Nafpliotis, N., y Goldberg, D. (1994). A niched pareto genetic algorithm for multiobjective optimization. En *Proceedings of the first ieee conference on evolutionary computation* (p. 82-87). NJ.

Kasuba, T. (1993, November). Simplified Fuzzy ARTMAP. *AI Expert*, 18–25.

Knowles, J., y Corne, D. (1999). The pareto archived evolution strategy: A new baseline algorithm for multiobjective optimization. En *Proceedings of the 1999 congress on evolutionary computation* (p. 98-105).

Minsky, M. (Ed.). (1988). *The society of mind*. First Touchstone Edition.

Srinivasan, D., y Rachmawati, L. (2006). An efficient multi-objective evolutionary algorithm with steady-state replacement model. En *Proceedings of the genetic and evolutionary computation conference (gecco-2006).*

Taghi, M., Baghmisheh, V., y Pavesic, N. (2003). A fast simpliffied fuzzy artmap network. *Neural Processing Letters*, *17*, 273–316.

Verzy, S. J., Heileman, G. L., Georgiopoulos, M., y Healy, M. H. (1998). Boosted ARTMAP. En *Proceedings of the international joint conference on neural networks* (Vol. 1, pp. 396–401). Anchorage, AK.

Williamson, J. R. (1996). Gaussian ARTMAP: A neural network for fast incremental learning of noisy multidimensional maps. *Neural Networks*, *9*(5), 881–897.

Williamson, J. R. (1997). A constructive, incremental-learning network for mixture modeling and classification. *Neural Computation*, *9*, 1517-1543.

Zeng, S., Yao, S., Kang, L., y Liu, Y. (2005). An efficient multiobjective evolutionary algorithm: OMOEA-II. En C. C. C. et. al. Eds (Ed.), *Emo 2005* (p. 108-119). Springer-Verlag.

Zitzler, E., Laumanns, M., y Thiele, L. (2002). SPEA2: Improving the Strength Pare-

to Evolutionary Algorithm for Multiobjective Optimization. En K. Giannakoglou y cols. (Eds.), *Evolutionary methods for design, optimisation and control with application to industrial problems (eurogen 2001)* (pp. 95–100). International Center for Numerical Methods in Engineering (CIMNE).

Zou, X., Liu, M., Kang, L., y He, J. (2004). A high performance multi-objective evolutionary algorithm based on the principles of thermodynamics. En X. Y. et al. Eds (Ed.), *Eighth int. conf. parallel problem solving from nature (ppsn viii)* (p. 922-931). Berlin, Germany: Springer-Verlag.

7. Anexos

7.1. Algoritmo *Fuzzy* ARTMAP

Algoritmo *Fuzzy* ARTMAP (propuesta original publicada en 1997) usando la versión en MATLAB 6.5 propuesta por el investigador Rajeev Raizada, PhD. tomando como base la propuesta original del CNS Technology LAB. El profesor Raizada ha colocado este material expresamente con fines académicos en el sitio de su laboratorio adscrito al Department of Brain and Cognitive Sciences at the University of Rochester (http://raizadalab.org/).

```
  %%% Implementation of a Fuzzy ARTMAP graphical demo, in Matlab
%%% Written by Rajeev Raizada, 16/4/97


disp('Graphical demo of Fuzzy ARTMAP learning');
disp('the Circle-in-the-Square task.');
disp('  ');
disp('Written by:  Rajeev Raizada, ');
disp('             Dept. of Cognitive and Neural Systems, ');
disp('             Boston University.');
disp('E.mail: rajeev@cns.bu.edu');
disp('Copyright (c) Rajeev Raizada, April 14th, 1997');
disp('  ');
disp('Original Reference:');
disp('Carpenter, G. A., Grossberg, S., Markuzon, N., ');
disp('Reynolds, J. H. and Rosen, D. B. (1992)        ');
disp('"Fuzzy ARTMAP: A Neural Network Architecture for Incremental');
disp('Supervised Learning of Analog Multidimensional Maps"');
disp('IEEE Transactions on Neural Networks,');
disp('Vol. 3, No. 5, pp. 698-713.');
```

```
disp('  ');
disp('Category boxes are shown, along with the');
disp('true decision boundary.');
disp('A category which resonates in the ARTa module for');
disp('a given input lights up, shown with a thick line.');
disp('It may then get reset by F_ab layer mismatch, and shown dotted.');
disp('Categories which match at F_ab can undergo learning.');
disp('Fast learning is used here, so a category box expands');
disp('to contain the input.');
disp('  ');
disp('Key:');
disp('Presented point:                                    *');
disp('Category which predicts being INSIDE the circle:  Solid line');
disp('Category which predicts being OUTSIDE the circle: Dashed line');
disp('Category resonating within ARTa module:           Thick line');
disp('Category reset by match-tracking:                 Dotted line');
disp('Point sized category: marked by a cross:          x');
disp('  ');
disp('After each step, the program pauses. PRESS ANY KEY TO CONTINUE');

colordef none; % So it looks ok on a black & white screen

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% The input and output

num_pats = 100;
sq = 1;                          % Size of square
r = sq/sqrt(2*pi);
```

```
                % Radius of circle so it's half area of square
xcent = 0.5;
ycent = 0.5;                    % Centre of circle
a = [ xcent*ones(1,num_pats); ycent*ones(1,num_pats)] + ...
        sq*(0.5-rand(2,num_pats));              % The x,y coords
bmat = ((a(1,:)-xcent).^2 + (a(2,:)-ycent).^2) > r^2;
bmat = [ bmat; 1-bmat ];        % Change to [1 0], [0 1] form

ac = [a; ones(size(a))-a];   % The complement-coded form of input a

%%%%%%%%%%%%%%%% If we are testing, make a grid of test inputs
test_mode = 0;    % Zero means we're testing, 1 means training

if test_mode==1,
        grain = 100;
        grx = linspace(xcent-sq/2,xcent+sq/2,grain)';
        gry = linspace(ycent-sq/2,ycent+sq/2,grain)';
        testpats = zeros(grain^2,2);    % Initialise
        for i=1:grain,
                testpats(1+grain*(i-1):i*grain,1) = grx;
                testpats(1+grain*(i-1):i*grain,2) = ...
                                        ones(grain,1)*gry(i);
        end;
        testpats = [ testpats' ; 1-testpats' ];
        test_out = zeros(size(testpats,2),1);
                                % Initialise output vector
end;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Parameters

alpha = 0.001;  % "Choice" parameter > 0. Set small for the
                % conservative limit (Fuzzy AM paper, Sect.3)
beta = 1;       % Learning rate. Set to 1 for fast learning
rho_a_bar = 0;  % Baseline vigilance for ARTa, in range [0,1]
M = size(a,1);  % Number of input components. Derived from data
                % NB: Total input size = 2M (due to complement)
N = 20;         % Number of available coding nodes
                % We start with some resonably large number
                % then, if we need to, can add more uncommitted
L = size(bmat,1);       % Number of output nodes. Derived from data
rho_ab = 0.95;          % Map field vigilance, in [0,1]
epsilon = 0.001;        % Fab mismatch raises ARTa vigilance to this
                        % much above what is needed to reset ARTa
num_patterns = size(a,2);   % Number of input patterns

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Set up weights

w_a = ones(2*M,N);      % Initial weights in ARTa. All set to 1
                        % Row-i, col-j entry = weight from input node i
                        % to F2 coding node j
w_ab = ones(L,N);       % Row-k, col-j entry = weight from ARTa F2
                        % node j to Map Field node k
committed_nodes = [];   % Keep a record of which nodes get committed
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Main loop
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for i=1:num_patterns    % Go through patterns one by one
                        % Note: could shuffle order using randperm

        A = ac(:,i);    % Present input is i-th column of ac

        %%%%%%%%%%%%%%%%%%% Plot all the category boxes
        figure(1);
        clf;
        %%%% Box x-path: u1, 1-vc1, 1-vc1, u1, u1
        %%%% Box y-path: u2, u2, 1-vc2, 1-vc2, u2
        hold on;
        h = plot([w_a(1,:); 1-w_a(3,:); 1-w_a(3,:); ...
                  w_a(1,:); w_a(1,:)], ...
                 [w_a(2,:); w_a(2,:); 1-w_a(4,:); ...
                  1-w_a(4,:); w_a(2,:)],'w-');
        %%%%%%%% Make the outside-circle predicting boxes dashed
        outboxes = find(w_ab(1,:)>w_ab(2,:));
                %%%%%%% w_ab will be near [1 0]
        if ~isempty(outboxes),
                set( h(outboxes),'LineStyle','--');
        end;
        %%%%%%%%%% Mark all the point-size boxes with crosses
        point_boxes = find(abs(sum(w_a)-M)<0.001);
        for j=1:length(point_boxes),
```

```
        hpoint = plot(w_a(1,point_boxes(j)), ...
                      w_a(2,point_boxes(j)),'x');
        h(point_boxes(j)) = hpoint;
              %%% Make handle point to 'x'
end;
%%%%%%%%%%%%%%%%%%%%%%%%%%% Plot the input vector
title(['Presenting input number ' num2str(i) ...
       '. It is ( ' num2str(A(1)) ...
       ', ' num2str(A(2)) ' )'  ]);
plot(A(1),A(2),'*');    % Show the input with a star
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Draw the circle
ang = 0:0.1:2.1*pi;
plot(xcent+r*cos(ang),ycent+r*sin(ang),'w-');
axis([-0.1 1.1 -0.1 1.1]);
axis('equal');
drawnow;
hold off;
pause;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Find an ARTa resonance which gives Fab resonance too

%%%%%%%%%%%%%%%%% Initialise
rho_a = rho_a_bar;
       % We start off with ARTa vigilance at baseline
resonant_a = 0;
       % We're not resonating in the ARTa module yet
```

```
        resonant_ab = 0;
                % Not resonating in the Fab match layer either
        already_reset_nodes = [];  % We haven't rest any ARTa nodes
                                   % for this input pattern yet

while resonant_ab == 0, %%%% We don't have an Fab match yet.
        %%%% So, we find an ARTa node and see what it predicts

        while resonant_a == 0,
                %%% In search of a resonating ARTa node...

                %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                %%%%%%%%%% Find the winning, matching ARTa node
                N = size(w_a,2);
                        % Count how many F2a nodes we have

                A_for_each_F2_node = A * ones(1,N);
                        % Matrix containing a copy of A for
                        % each F2 node. Useful for Matlab

                A_AND_w = min(A_for_each_F2_node,w_a);
                        % Fuzzy AND = min

                S = sum(A_AND_w);
                        % Row vector of signals to F2 nodes

                T = S ./ (alpha + sum(w_a));
```

```
        % Choice function vector for F2

%%%%%%%%% Set all the reset nodes to zero

T(already_reset_nodes) = ...
        zeros(1,length(already_reset_nodes));

%%%%%%%%%%%% Finding the winning node, J

[ Tmax, J ] = max(T);
        % Matlab function max works such that
        % J is the lowest index of max T elements, as
        % desired. J is the winning F2 category node

y = zeros(1,N);
y(J)=1;
        % Activities of F2. All zero, except J

w_J = w_a(:,J);
        % Weight vector into winning F2 node, J

x = min(A,w_J);
        % Fuzzy version of 2/3 rule. x is F1 activity
        % NB: We could also use J-th element of S
        % since the top line of the match fraction
        % |I and w|/|I| is sum(x), which is
        % S = sum(A_AND_w) from above
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%% Testing if the winning node resonates in ARTa

if sum(x)/sum(A) >= rho_a,
                          % If a match, we're done
        resonant_a = 1;          % ARTa resonates
end;
% The while resonant_a == 0 command will stop looping
% now, so we exit the while loop and go onto to Fab

if sum(x)/sum(A) < rho_a,
        % If mismatch then we reset
        resonant_a = 0;     % So, still not resonating
        already_reset_nodes = ...
                          [ already_reset_nodes J ];
end;            % Record that node J has been
                % reset already.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Creating a new node if we've reset all of them

if length(already_reset_nodes) == N,
        % If all F2a nodes reset
        w_a = [ w_a ones(2*M,1) ];
        % Add uncommitted node to ARTa weight vector
        w_ab = [w_ab ones(L,1) ];
```

```
                % Give the new F2a node a w_ab entry
                % Draw on the new uncommitted category box
                hold on;
                hnew = plot([0 1 1 0 0],[0 0 1 1 0],'w-');
                hold off;
                h = [ h; hnew ];
                %%% Add handle for new box to list
        end;    % Now go back and this new node should win


end;    % End of the while loop searching for ARTa resonance
        % If resonant_a = 0, we pick the next highest Tj
        % and see if *that* node resonates, i.e. goto "while"
        % If resonant_a = 1, we have found an ARTa resonance,
        % namely node J
        % So we go on to see if we get Fab match with node J

%%%%%%%%% Show the winning, matching ARTa node J
set(h(J),'LineWidth',[4]);
title(['Node ' num2str(J) ' was chosen by ARTa search.' ...
        ' Choice = ' num2str(Tmax) '. Match = ' ...
        num2str(sum(x)/sum(A)) '. Rho_a = ' num2str(rho_a) ]);
pause;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Match tracking

if test_mode==0,                % If we're not testing
```

```
        b = bmat(:,i);
                % Desired output for input number i
        z = min(b,w_ab(:,J));   % Fab activation vector, z
                % (Called x_ab in Fuzzy ARTMAP paper)

        %%%%%% Test for Fab resonance

        if sum(z)/sum(b) >= rho_ab,     % We have an Fab match
                resonant_ab = 1;
                title(['Desired: ' num2str(b') ...
'. F_{ab} act: ' num2str(z') '.  F_{ab} match score: ' ...
num2str(sum(z)/sum(b)) '. Rho_{ab} = ' num2str(rho_ab) ...
'. Match field resonates.']);
                pause;
        end;            % This will cause us to leave the
                        % while resonant_ab==0 loop and
                        % go on to do learning.

        if sum(z)/sum(b) < rho_ab,
                        % We have an Fab mismatch
                resonant_ab = 0;
                        % This makes us go through
                        % the resonant_ab==0 loop again
                resonant_a = 0; % This makes us go through
                                % ARTa search again, this
                                % search being inside the
                                % resonant_ab==0 loop
```

```
        % Increase rho_a vigilance.
        % This will cause F2a node J to get reset when
        % we go back through the ARTa search loop again.
        % Also, *for this input*, the above-baseline
        % vigilance will cause a finer ARTa category to win

                rho_a = sum(x)/sum(A) + epsilon;

                %%%% Make the reset box dotted and thin
                title(['Desired out: ' num2str(b') ...
'. F_{ab} act: ' num2str(z') '. Match field resets. ' ...
'Rho_a increased to ' num2str(rho_a) ]);
                set(h(J),'LineStyle',':');
                set(h(J),'LineWidth',[0.5]);
                pause;

        end;    % End of Fab mismatch if

 end;           % End of "if test_mode==0"

 if test_mode==1,
        % If we're testing there can be no Fab mismatch
        % We just want the prediction.
        resonant_ab = 1;  % This is just to exit the loop
        test_out(i) = w_ab(:,J)/sum(w_ab(:,J));
        % The prediction is the normalised w_ab vector
 end;
```

```
end;    %%%%%%%%% End of the while resonant_ab==0 loop.
        %%%%%%%%% Now we have a resonating ARTa output
        %%%%%%%%% which gives a match at the Fab layer.
        %%%%%%%%% So, we go on to have learning
        %%%%%%%%% in the w_a and w_ab weights


        %%%%%%%%%% Let the winning, matching node J learn

        w_a(:,J) = beta*x + (1-beta)*w_a(:,J);
                        % NB: x = min(A,w_J) = I and w
                        %%%% Learning on F1a <--> f2a weights

        w_ab(:,J) = beta*z + (1-beta)*w_ab(:,J);
                        % NB: z=min(b,w_ab(J))=b and w

end;    %%%%%%%%% End of considering this input.
        %%%%%%%%% Now we go on to the next input, and repeat
```

7.2. Algoritmo FS-FAM modificado a partir del *Fuzzy* ARTMAP

```
%%% Implementation of FS-FAM ARTMAP in Matlab

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% The input and output

   clear;
load Base10000.mat;

%%%%%%%%%%%%%%%%% If we are testing, make a grid of test inputs
test_mode = 0;    % Zero means we're training, 1 means testing

if test_mode==1,
        grain = 100;
        grx = linspace(xcent-sq/2,xcent+sq/2,grain)';
        gry = linspace(ycent-sq/2,ycent+sq/2,grain)';
        testpats = zeros(grain^2,2);    % Initialise
        for i=1:grain,
                testpats(1+grain*(i-1):i*grain,1) = grx;
                testpats(1+grain*(i-1):i*grain,2) = ...
                                        ones(grain,1)*gry(i);
        end;
        testpats = [ testpats' ; 1-testpats' ];
        test_out = zeros(size(testpats,2),1);
                                % Initialise output vector
end;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Parameters
```

```
alpha = 0.001;  % "Choice" parameter > 0. Set small for the
                % conservative limit (Fuzzy AM paper, Sect.3)
beta = 1;       % Learning rate. Set to 1 for fast learning
rho_bar = 0;    % Baseline vigilance for ARTa, in range [0,1]
M = size(a,1);  % Number of input components. Derived from data
                % NB: Total input size = 2M (due to complement)
N = 1;         % Number of available coding nodes
                % We start with some resonably large number
                % then, if we need to, can add more uncommitted
L = size(bmat,1);       % Number of output nodes. Derived from data
epsilon = 0.001;        % Fab mismatch raises ARTa vigilance to this
                        % much above what is needed to reset ARTa
num_patterns = size(a,2);   % Number of input patterns

class_idx = 2*M+1;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Set up weights

w = [ones(2*M,1); -1];  % Initial weights in ARTa. All set to 1,
                        % classification is none = -1 uncommitted node
                        % Row-i, col-j entry = weight from input node i
                        % to F2 coding node j

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Main loop
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
tic
for i=1:num_patterns    % Go through patterns one by one
                        % Note: could shuffle order using randperm

        A = ac(:,i);    % Present input is i-th column of ac

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %%%%%%%%%% Find the winning, matching ARTa node

        N = size(w,2);
        % Count how many F2a nodes we have

        A_for_each_F2_node = A * ones(1,N);
        % Matrix containing a copy of A for
        % each F2 node. Useful for Matlab

        A_AND_w = min(A_for_each_F2_node,w(1:2*M,:));
        % Fuzzy AND = min

        S = sum(A_AND_w);
        % Row vector of signals to F2 nodes

        %%%%%%%%%%%%%%%% Initialise
        rho = rho_bar;
        % We start off with ARTa vigilance at baseline
        resonant = 0;
        % We're not resonating in the ARTa module yet
```

```
        rho_vect = S / sum(A);
        w_pass   = w(:,find(rho_vect >= rho));
        w_not    = w(:,find(rho_vect < rho));
        rho_pass = rho_vect(find(rho_vect >= rho));

        while (~resonant)
          T = rho_pass * sum(A) ./ (alpha+sum(w_pass));
          [ Tmax, J ] = max(T);
          if (w_pass(class_idx,J) == -1) % uncommited node
            w_pass = [w_pass, [A; bmat(i)]];
            resonant = 1;
          elseif (w_pass(class_idx,J) == bmat(i))
            w_pass(1:2*M,J) = (1-beta)*w_pass(1:2*M,J) + beta* ...
                min(w_pass(1:2*M,J), A);
            resonant = 1;
          else
            % matchtracking
            rho      = rho_pass(J) + epsilon;
            w_not    = [w_not w_pass(:,find(rho_pass < rho))];
            alive    = find(rho_pass >= rho);
            w_pass   = w_pass(:,alive);
            rho_pass = rho_pass(alive);
          end
        end
        w = [w_pass w_not];
end
```

```
[SO sort_order] = sort(w(1,:));
w = w(:,sort_order);
disp('iteration: '), disp(i);
disp(w);
disp('FS-FAM finished.');
toc
%end
```

7.3. Modificación propuesta Pareto FAM

```
%%% Implementation of Pareto FAM  in Matlab

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% The input and output

clear;
load Base10000.mat;

%%%%%%%%%%%%%%%%% If we are testing, make a grid of test inputs
test_mode = 0;    % Zero means we're training, 1 means testing

if test_mode==1,
        grain = 100;
        grx = linspace(xcent-sq/2,xcent+sq/2,grain)';
        gry = linspace(ycent-sq/2,ycent+sq/2,grain)';
        testpats = zeros(grain^2,2);    % Initialise
        for i=1:grain,
                testpats(1+grain*(i-1):i*grain,1) = grx;
                testpats(1+grain*(i-1):i*grain,2) = ...
                                        ones(grain,1)*gry(i);
        end;
        testpats = [ testpats' ; 1-testpats' ];
        test_out = zeros(size(testpats,2),1);
                                % Initialise output vector
end;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Parameters
```

```
alpha = 0.001;  % "Choice" parameter > 0. Set small for the
                % conservative limit (Fuzzy AM paper, Sect.3)
beta = 1;       % Learning rate. Set to 1 for fast learning
rho = 0;        % Baseline vigilance for ARTa, in range [0,1]
M = size(a,1);  % Number of input components. Derived from data
                % NB: Total input size = 2M (due to complement)
N = 1;         % Number of available coding nodes
                % We start with some resonably large number
                % then, if we need to, can add more uncommitted
L = size(bmat,1);       % Number of output nodes. Derived from data
epsilon = 0.001;        % Fab mismatch raises ARTa vigilance to this
                        % much above what is needed to reset ARTa
num_patterns = size(a,2);   % Number of input patterns

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Set up weights

w = [ones(2*M,1); -1];  % Initial weights in ARTa. All set to 1,
                        % classification is none = -1 uncommitted node
                        % Row-i, col-j entry = weight from input node i
                        % to F2 coding node j

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Main loop
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
tic
for i=1:num_patterns    % Go through patterns one by one
                        % Note: could shuffle order using randperm
```

```
A = ac(:,i);    % Present input is i-th column of ac

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%% Find the winning, matching ARTa node

N = size(w,2);
% Count how many F2a nodes we have

A_for_each_F2_node = A * ones(1,N);
% Matrix containing a copy of A for
% each F2 node. Useful for Matlab

A_AND_w = min(A_for_each_F2_node,w(1:2*M,:));
% Fuzzy AND = min

S = sum(A_AND_w);
% Row vector of signals to F2 nodes

rho_vect = S / sum(A);



candidate_indexes = find(rho_vect >= rho);
w_not    = w(:,find(rho_vect < rho));

w_pass   = w(:,candidate_indexes);
rho_vect = rho_vect(:,candidate_indexes);
```

```
T_vect = S(:,candidate_indexes) ./ (alpha+sum(w_pass));

rho_T_vect = [rho_vect; T_vect];
pos = 1;

for pos = 1:size(rho_T_vect,2)
  if (rho_T_vect(1,pos) > 0) && (rho_T_vect(2,pos) > 0)
    smallers = [rho_T_vect(1,:) < rho_T_vect(1,pos) ; rho_T_vect(
    indexes  = find(sum(smallers) == 2);
    rho_T_vect(:,indexes) = 0;
  end
end

pareto_indexes = find(sum(rho_T_vect) > 0);
non_pareto     = find(sum(rho_T_vect) == 0);
pareto   = rho_T_vect(:, pareto_indexes);
w_pareto = w_pass(:,pareto_indexes);
w_not_pareto = w_pass(:,non_pareto);

[SO sorted_indexes] = sort(pareto(1,:));
pareto = pareto(:,sorted_indexes);
w_pareto = w_pareto(:,sorted_indexes);

J = 1;
class = 2*M+1;
while w_pareto(class,J) ~= -1 && w_pareto(class,J) ~= bmat(i)
  J = J+1;
```

```
        end

        if (w_pareto(class,J) == -1) % uncommited node
          w_pareto = [w_pareto, [A; bmat(i)]];
        else % if (w_pareto(class,J) == bmat(J))
          w_pareto(1:2*M,J) = (1-beta)*w_pareto(1:2*M,J) + beta*min(w_par
        end

        w = [w_pareto w_not_pareto w_not];
end

[SSO sort_order] = sort(w(1,:));
w = w(:,sort_order);
disp('iteration: '), disp(i);
disp(w);
disp('Pareto FAM finished');
toc
```

Para crear las bases se utilizaron las instrucciones:

```
num_pats=1000;  %100, 1000, 10000
sq = 1;                          % Size of square
r = sq/sqrt(2*pi);               % Radius of circle so it's half area o
xcent = 0.5;
ycent = 0.5;                     % Centre of circle
a = [ xcent*ones(1,num_pats); ycent*ones(1,num_pats)] + sq*(0.5-rand(2
bmat = ((a(1,:)-xcent).^2 + (a(2,:)-ycent).^2) > r^2;
```