Universidad de la República
Facultad de Ingeniería

Algoritmo SOM en Sistemas Embebidos y su aplicación como ANTISPAM

Autores
Carlos Laicovsky Jitar
Rodrigo San Vicente Manco

Tutor
Dr. Ing. Enrique Ferreira

Mayo 2007

Proyecto en Ingeniería Eléctrica Plan 97

Algoritmo SOM en Sistemas Embebidos y su aplicación como ANTISPAM

Carlos Laicovsky Jitar, Rodrigo San Vicente Manco

Resumen

El presente proyecto realiza una aproximación a la técnica *SOM* de aprendizaje automático por redes neuronales, la cual cobra gran relevancia en la actualidad, dado que es la base de muchas herramientas de *data mining*. Se propone la implementación de *SOM* en una nueva arquitectura, en el campo de los sistemas embebidos, que combina velocidad, paralelismo y bajo costo de desarrollo. En base a esta plataforma se desarrolla una solución basada en *machine learning* al problema de detección de *spam*.

La plataforma elegida es el *DSP* **Blackfin bf537** basado en **uClinux**, variante de *Linux* para sistemas embebidos. La herramienta antispam desarrollada se integra al cliente de correo **Evolution** de *Linux*, no siendo compleja su adaptación a otros clientes. La solución es fácilmente escalable a las necesidades de detección de *spam* de una red corporativa.

Se obtuvo un buen resultado en la implementación de *SOM* sobre el *DSP* tanto en entrenamiento como en clasificación. Dado el carácter *opensource* del entorno de desarrollo en el *DSP* se adquirió un conocimiento importante en la programación en ambiente *Linux*. Las comparaciones de *performance* realizadas con otras herramientas antispam disponibles comercialmente muestran resultados promisorios, con importantes mejoras en cuanto a velocidad de clasificación. Operando el hardware en forma *standalone* se obtiene una alternativa de bajo costo frente a un servidor centralizado para la protección contra el *spam*.

C.Laicovsky, e-mail: claico@adinet.com.uy
R.San Vicente, e-mail: rodsvm@gmail.com

Índice

I. Introducción **1**

A. Planteamiento del problema 1

B. Objetivos 4

C. Organizacion del documento 4

II. Redes Neuronales **6**

A. Introducción 6

B. Modelado Neuronal 6

B.1. Elementos internos de una red 7

B.2. Caracteristicas de una red 10

B.3. Aprendizaje 11

C. Redes Neuronales SOM 13

C.1. Origen fisiológico del modelo SOM 13

C.2. Arquitectura 15

C.3. Modelo básico SOM 16

C.4. Implementación de un clasificador basado en SOM 19

D. Antecedentes de implementación 20

III. Spam **22**

A. Introducción 22

B. Terminología 22

C. Análisis del problema 24

D. Características del *spam* 26

IV. Descripción general del sistema **28**

A. Introducción 28

B. Descripción de bloques de software 28

C. Componentes del sistema 28

C.1. Bloques de Hardware 28

C.2. Bloques de Software 29

C.3. Plataforma de Desarrollo 30

C.4. Canales de Comunicación 30

V. Elección de Hardware **31**

A. Introducción 31

B. Analisis de distintas alternativas ofrecidas por el mercado 31

B.1. Arquitecturas DSP 31

B.2. Otras Arquitecturas de Procesadores para sistemas embebidos 36

B.3. Criterios de selección 39

C. Plataforma adoptada 40

VI. Adaptación del sistema al hardware disponible **42**

A. Introducción 42

B. Características del hardware disponible 42

C. Entorno de desarrollo 44

D. Ajustes realizados al *kernel* 47

E. Paquete SOM_PAK 48

F. Adaptación de la aplicación a la placa 49

F.1. Evaluación de resultados 56

VII. Diseño de la aplicación anti *SPAM* 58

A. Introducción . . . 58

B. Arquitectura de la solución propuesta . . . 58

B.1. Solución adoptada . . . 61

C. Implementación de la solución . . . 63

C.1. Procesos en el Cliente de Correo . . . 64

C.2. Procesos en el *Servidor SOM* . . . 64

D. Descripción de los bloques de software de la solución . . . 64

D.1. Programa de codificación de los mensajes del cliente: *trainsom* . . . 65

D.2. Programa de envío de la codificación al *Servidor SOM*: *somasspam* . 66

D.3. Sistema ejecutandose en el *Servidor SOM* remoto: *som* . . . 66

D.4. Parámetros inherentes a la estructura de la red neuronal . . . 67

D.5. Parámetros inherentes a la funcionalidad anti *spam* . . . 68

E. Notas sobre el diseño . . . 68

E.1. Codificación de los mensajes . . . 69

E.2. Almacenamiento de datos - Formato de los Archivos . . . 71

E.3. Cálculo del *Score de Spam* . . . 77

E.4. Ejemplo de cálculo de *Score de Spam* . . . 80

E.5. Archivos de Programas . . . 82

E.6. Comunicación de la aplicación anti *spam* con la red *som* . . . 85

E.7. Mejoras a la interfaz de usuario . . . 86

E.8. Integración a otras herramientas de correo . . . 86

VIII. Comparación con otras herramientas anti *spam* 88

A. Introducción . . . 88

B. Comparación de tiempos de entrenamiento 88
C. Comparación del desempeño y de la eficacia en la detección 89
D. Comparación en el manejo de la aplicación 90

IX. Conclusiones **95**
A. Introducción 95
B. Conclusiones generales 95
C. Líneas de trabajo futuro 97

Referencias **99**

A. Apéndice: Descripción del procesador Blackfin **102**
A. Introducción 102
B. Micro Signal Architecture 102
C. Familia Blackfin 103

B. Apéndice: Contenido del CD adjunto **109**
A. Introducción 109
B. Estructura de directorios 109

Índice de figuras

1. Elementos internos de una red neuronal 7
2. Red *feedforward* de varias capas 9
3. Red SOM de m entradas y l salidas 14
4. Proceso de competición 17
5. Diagrama de Bloques del Sistema 28
6. Diagrama de Bloques de Software del Sistema 29
7. Placa *bf537 stamp* 42
8. *uClinux* para *Blackfin* 44
9. Configuración de filtros en *Evolution* 59
10. Técnicas de Análisis de Correo Electrónico 61
11. Diagrama General del Sistema 63
12. Proceso *trainsom* 65
13. Proceso *somasspam* 65
14. Mapa SOM entrenado para la detección de *spam* 76
15. Neurona ganadora y su entorno en el Mapa SOM 77
16. Neurona ganadora y su posición en el Mapa SOM 81
17. Integración con otras herramientas de *spam* en *Evolution* 87
18. Configuración de spamassassin en *Evolution* 91
19. Configuración de *trainsom -dh (somasspam)* en *Evolution* 92
20. Selección de mails en *Evolution* 93
21. Configuración de *somasspam* en *Evolution* 94
22. Estructura Interna del procesador *Blackfin* 105
23. Mapa de memoria del procesador *bf537* 107

Índice de cuadros

I. Selección de DSP 35

II. Fabricantes y Modelos de microprocesadores 40

III. Plataforma Adoptada 41

IV. Formato de punto flotante precisión simple en *Flat ELF* 52

V. Comparación de tiempos de entrenamiento 57

VI. Tiempos de entrenamiento *spamassasin* vs *somasspam* 88

VII. Comparación de desempeño y eficacia *spamassassin* vs *somasspam* 89

I. Introducción

A. Planteamiento del problema

Existe actualmente una tendencia a establecer un nuevo campo de las ciencias de la computación. Su objetivo es integrar los diferentes métodos de resolución de problemas que no pueden ser descritos fácilmente mediante un enfoque algorítmico tradicional. Estos métodos, de una forma u otra, tienen su origen en la emulación, más o menos inteligente, del comportamiento de los sistemas biológicos.

Dentro de dicha tendencia se encuentran las Redes Neuronales Artificiales (ANN), técnicas en las cuales se han logrado importantes avances que imitan el funcionamiento del sistema nervioso para la resolución de problemas. A través de procesos de "aprendizaje" los sistemas basados en ANN son capaces de manejar las imprecisiones e incertidumbres que aparecen cuando se trata de resolver problemas relacionados con el mundo real (reconocimiento de formas, toma de decisiones, etc.).

Con las Redes Neuronales se intenta expresar la solución de problemas complejos, no como una secuencia de pasos, sino como la evolución de sistemas de computación inspirados en el funcionamiento del cerebro humano, y dotados por tanto de cierta "inteligencia", los cuales no son sino la combinación de una gran cantidad de elementos simples de proceso (neuronas) interconectados que, operando de forma masivamente paralela, consiguen resolver problemas relacionados con el reconocimieto de formas o patrones, predicción, codificación, clasificación, control y optimización.

En este sentido, es importante destacar que, si bien se desarrollan aplicaciones de ANN mediante programas de simulación, cabe esperar que todo el potencial de las redes neuronales se ponga de manifiesto precisamente mediante su implementacion física en hardware específico. Se ven ventajas en explotar el aprendizaje con multiples elementos de proceso que

permita explotar el paralelismo inherente a las redes neuronales, frente a la codificación en hardware estándar de algoritmos de funcionamiento

El presente proyecto trata sobre el modelo de red SOM (Self Organizing Map), ideado por Teuvo Kohonen, basado en la capacidad inherente del cerebro de formar "mapas topológicos" de las informaciones recibidas del exterior. En este sentido, la red analizada es capaz de organizar por sí misma mapas que reflejan las características comunes de las informaciones aprendidas, garantizando que las neuronas topológicamente próximas sean sensibles a entradas físicamente similares.

Tradicionalmente las implementaciones en hardware de ANN han sido las siguientes:

- Diseño de integrados específicos que implementen ANN (ASIC[1]). Durante la década de los '90 se han desarrollado infinidad de integrados que implementan ANN, aunque han sido caros y han tenido poco éxito comercial. En su mayoría están discontinuados.
- Implementación de ANN en hardware configurable del tipo FPGA[2]. Este tipo de soluciones no suelen escalar en costos para la producción masiva.

En el presente proyecto se propone estudiar una aproximación intermedia: implementar la red SOM en un DSP (Procesador Digitales de Señales). La arquitectura de este tipo de procesadores implementa paralelismo en forma limitada, con la gran ventaja que su costo hace posible el disponer de una solución óptima para la producción en gran escala.

Las plataformas DSP de última generación están pensadas para implementar soluciones de sistemas embebidos, los cuales son sistemas supervisados por una computadora hecha para cumplir un cometido específico, a diferencia de los sistemas de proposito general. Muchos de estos DSP pasan a tener funciones de Microcontrolador, aumentando su adaptabilidad a diversos tipos de problemas.

[1] Application Specific Integrated Circuit
[2] Field Programmable Gate Array

Para dotar de flexibilidad a este tipo de plataformas se incorpora, por lo general un sistema operativo que permite su adaptación a los requerimientos de las distintas aplicaciones. Desde el punto de vista económico es fundamental que los costos las herramientas de desarrollo sean lo mas razonables posibles

Por ello están tomando fuerza en los últimos tiempos las plataformas basadas en sistema operativo Linux embebido, las cuales tienen herramientas de desarrollo Open Source gratuitas en su mayoría.

Se va a hacer énfasis en seleccionar una plataforma embebida de bajo costo y alta performance, en principio un DSP, aunque se explorarán también otras arquitecturas a los efectos de comparar capacidades.
Toda esta implementación se aplicará, en el presente proyecto, al problema concreto de la detección de *spam* en correos electrónicos.

La definición mas general de *spam* consiste en correo anónimo, masivo y no solicitado. En contrapartida el correo legitimo se denomina *ham*. Ahora bien, puede existir correo legitimo no solicitado, como por ejemplo propuestas comerciales, consideradas como validas por el destinatario resultando ser *ham*.

La detección de *spam* pasa a ser un problema personal ya que la clasificación de un correo como *spam* depende del usuario del email. Un correo que es *spam* para una persona puede no serlo para otra. A su vez los mensajes de *spam* evolucionan en su técnica, y puede ser que herramientas que anteriormente eran efectivas para detectar el *spam* ya no lo sean para los *spams* generados en la actualidad.

Por ello se necesita un mecanismo que aprenda cual es la definición de *spam* para el dueño de la casilla de email, la aplique a los correos venideros para clasificar los mismos y adapte dicha definición a las preferencias futuras del usuario.

En conclusión se considera que este problema está particularmente bien adaptado al empleo de redes neuronales para su resolución, en particular la red neuronal SOM, dado que la misma tiende a agrupar estimulos similares en neuronas próximas

B. Objetivos

- Estudiar el mercado mundial de posibles procesadores de sistemas embebidos, poniendo enfasis en costo, performance, costo de entorno de desarrollo, etc.
- Familiarizarse con las técnicas de desarrollo en linux, aplicadas en sistemas embebidos
- Lograr una plataforma hardware SOM con buena relación performance-costo respecto a las plataformas de hardware genérico.
- Construir una aplicación *antispam* basada en la plataforma SOM que pueda ser usada tanto en una PC de hogar como en una red corporativa.
- Comparar la herramienta desarrollada con herramientas similares de uso corriente.

C. Organizacion del documento

En el Capítulo II se incluye una reseña teórica acerca de los conceptos de redes neuronales involucrados en el proyecto, con énfasis particular en SOM.

En el Capítulo III es un repaso al concepto de *spam*.

En el Capítulo IV se describe la forma general del sistema a desarrollar.

En el Capítulo V se habla acerca del proceso de investigación relacionado con la elección de una plataforma de sistemas embebidos para el desarrollo.

En el Capítulo VI está la descripción de las acciones realizadas para implementar SOM en la plataforma elegida.

En el Capítulo VII se describe la solución *antispam* adoptada.

En el Capítulo VIII se compara el sistema con otras herramientas *antispam*.

En el Capítulo IX se plantean las conclusiones y futuras líneas de investigación.

Finalmente, se incluye la lista de referencias que fueron de utilidad en el proyecto.

II. Redes Neuronales

A. Introducción

Este capítulo comienza con una introducción al tema de las redes neuronales artificiales (ANN[3]), siguiendo con una exposición de los conceptos teóricos del modelo de mapas autoorganizados (SOM[4]) de Kohonen, detallando sus aplicaciones y las implementaciones realizadas.

B. Modelado Neuronal

Konohen [1] expone la siguiente definición de ANN: "Las redes neuronales artificiales son redes interconectadas masivamente en paralelo de elementos simples (usualmente adaptativos) y con organización jerárquica, las cuales intentan interactuar con los objetos del mundo real del mismo modo que lo hace el sistema nervioso biológico".

El enfoque de resolución de problemas por medio de redes neuronales artificiales presentan las siguientes ventajas respecto a los sistemas algoritmicos tradicionales:

- Aprendizaje adaptativo. Es la capacidad de aprender a realizar tareas basada en un entrenamiento o una experiencia inicial.
- Autoorganización. Es el proceso de aprendizaje mediante el cual los enlaces ponderados de las neuronas se ajustan de manera que se obtengan resultados especificos. Una red neuronal puede crear su propia organización o representación de la información que recibe mediante una etapa de aprendizaje.
- Tolerancia a fallos. La destrucción parcial de una red conduce a una degradación de su estructura; sin embargo, algunas capacidades de la red se pueden retener, incluso sufriendo un gran daño.

[3] Artificial Neural Networks
[4] Self Organazing Map

- Operación en tiempo real. Los computadores neuronales pueden implementarse en paralelo, y se diseñan y fabrican máquinas con hardware especial para obtener esta capacidad.

Las redes neuronales son modelos que intentan reproducir el comportamiento del cerebro. Como tal modelo realiza una simplificación, averiguando cuáles son los elementos relevantes del sistema, debido a que la cantidad de información de que se dispone es excesiva o es redundante.

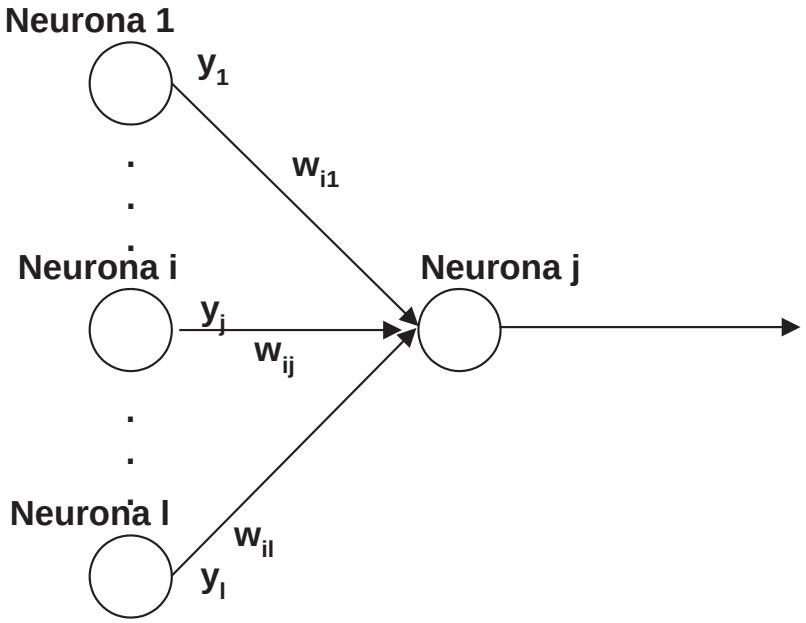

Figura 1. Elementos internos de una red neuronal

B.1 Elementos internos de una red

Los elementos de una red neuronal artificial (ver figura 1) son los siguientes:

- Unidades de proceso: La neurona artificial. Su trabajo es simple y único. Consiste en recibir las entradas de las células vecinas y calcular un valor de salida, el cual es enviado a las otras unidades.
- Estado de activación. Todas las neuronas que componen la red se hallan en cierto

estado. En una visión simplificada, podemos decir que hay dos posibles estados, reposo y excitado, a los que denominaremos globalmente estados de activación $a(t)$ y a cada uno de los cuales se le asigna un valor.

- Función de salida o de transferencia. Entre las unidades o neuronas que forman una red neuronal artificial existe un conjunto de conexiones que unen unas a otras. Cada unidad transmite señales a aquellas que están conectadas con su salida. Asociada con cada unidad o neurona hay una función de salida $f(a(t))$, que transforma el estado actual de activación $a(t)$ en una señal de salida $y(t)$,es decir: $y(t) = f(a(t))$. Existen varias posibles funciones de transferencia:
  - Identidad.
  - Escalón.
  - Sigmoidal.
  - Gaussiana.
- Conexiones entre neuronas. Las conexiones que unen a las neuronas que forman una ANN tienen asociado un peso, que es el que hace que la red adquiera conocimiento. Consideremos y_i como el valor de salida de una neurona i en un instante dado. Una neurona recibe un conjunto de señales que le dan información del estado de activación de todas las neuronas con las que se encuentra conectada. Cada conexión (denominada sinapsis) entre la neurona i y la neurona j está ponderada por un peso sináptico w_{ij}. Las entradas que llegan a una neurona se combinan entre si, formando lo que se denomina entrada neta de la red: $Net_j = \sum_i y_i w_{ji}$
- Función de activación. Así como es necesario una regla que combine las entradas a una neurona con los pesos de las conexiones, también se requiere una regla (F) que combine las entradas con el estado actual de la neurona para producir un nuevo estado

de activación. Es decir: $a_j(t+1) = F(a_j(t), Net_j)$ Por lo general: $F(a_j(t), Net_j) = Net_j - \theta_j$ siendo θ el umbral de activación de la neurona, el cual puede llegar a ser diferente para cada unidad. Todo esto implica que la salida de la neurona es: $y(t+1) = f(Net_j - \theta_j)$

- Regla de aprendizaje. Consiste en la modificación del comportamiento inducido por la interacción con el entorno y como resultado de experiencias conducente al establecimiento de nuevos modelos de respuesta a estímulos externos. En el caso de las redes neuronales artificiales, se puede considerar que el conocimiento se encuentra representado en los pesos de las conexiones entre neuronas. Todo proceso de aprendizaje implica cierto número de cambios en estas conexiones. Puede decirse que se aprende modificando los valores de los pesos de la red.

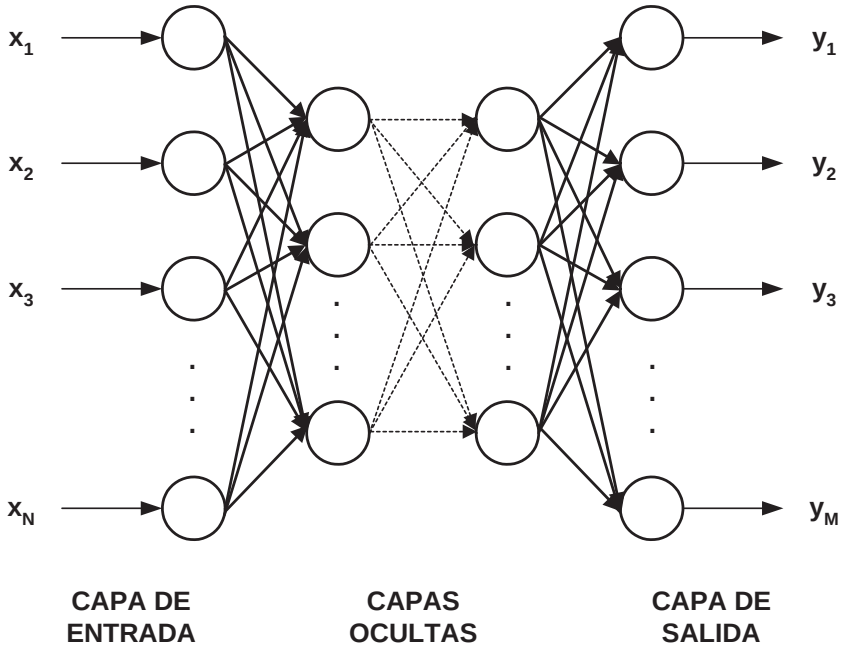

Figura 2. Red *feedforward* de varias capas

B.2 Caracteristicas de una red

La topología de una red neuronal esta caracterizada por lo siguiente:

- Capas de neuronas. La distribución de neuronas dentro de la red se realiza formando niveles o capas de un número determinado de neuronas cada una. Existen tres tipos de capas:
  1. Entrada. Las neuronas de la capa reciben información de fuentes externas a la red.
  2. Oculta. La capa no tiene contacto con el exterior, siendo interna a la red.
  3. Salida. Transfiere la información de la red hacia el exterior.
- Número de neuronas por capa. La cantidad de neuronas por nivel caracteriza a la red.
- Patrones de conexión entre neuronas. La conectividad entre los nodos de una red neuronal está relacionada con la forma en que las salidas de las neuronas se convierten en entradas de otras neuronas. Existen los siguientes tipos:
  1. Conexión hacia adelante. Las salidas de las neuronas son entrada de neuronas de niveles posteriores.
  2. Conexión hacia atrás. Las salidas pueden ser conectadas como entradas de neuronas de niveles precedentes.
  3. Conexión autorecurrente. La salida de la neurona es aplicada a su propia entrada.
  4. Conexión lateral. Una neurona de una capa determinada influye sobre otras neuronas de su misma capa.
- Flujo de información. Se distinguen dos tipos de redes:
  1. Red *feedforward*. Todas las señales se propagan hacia adelante en las capas de la red, sin conexiones hacia atrás, pero eventualmente con algún tipo de propagación lateral débil.

2. Red *feedforward/feedback*. Este tipo de redes presentan alguna forma de conexión hacia atrás y/o recurrente, además de otro tipo de conexiones. En general suelen ser de dos capas, trabajando en resonancia, es decir las dos capas interactúan entre si hasta llegar a un estado estable.

B.3 Aprendizaje

El aprendizaje es el proceso por el cual una red neuronal modifica sus pesos en respuesta a una información de entrada. Los cambios que se producen durante el proceso de aprendizaje se reducen a la destrucción, modificación y creación de conexiones entre las neuronas.

Durante el proceso de aprendizaje, los pesos de las conexiones de la red sufren modificaciones, por tanto se puede afirmar que este proceso ha terminado (la red ha aprendido) cuando los valores de los pesos permanecen estables. Es decir la red se considera entrenada cuando $\frac{dw_{ij}}{dt} = 0$

Un aspecto importante respecto al aprendizaje en las redes neuronales es el conocer cómo se modifican los valores de los pesos; es decir, cuáles son los criterios que se siguen para cambiar el valor asignado a las conexiones cuando se pretende que la red aprenda una nueva información. Estos criterios determinan lo que se conoce como la regla de aprendizaje de la red.

Existen los siguientes tipos de aprendizaje:

- Aprendizaje supervisado. El proceso de aprendizaje se realiza mediante un entrenamiento controlado por un agente externo, denominado "experto", que determina la respuesta que debería generar la red a partir de una entrada determinada. El experto comprueba la salida de la red y en caso que la misma no coincida con la deseada se procede a modificar los pesos de las conexiones.
- Aprendizaje no supervisado. La red no recibe ninguna información externa para ajus-

tar sus pesos. Esta caracteristica se denomina autoorganización de la red. Dependiendo de la estructura y el tipo de algoritmo de aprendizaje, se puede obtener lo siguiente de los datos de entrada:

- Codificación de los datos de entrada. Manteniendo la información relevante de la entrada, se genera una salida, en la red neuronal, con menor cantidad de bits que la entrada.
- Mapeo de características. La neuronas de salida presentan una disposición geométrica que constituye un mapa topográfico de las características de los datos de entrada, tales que datos de entrada con características similares afecten neuronas en la misma zona del mapa. Este mapeo conduce a una clasificación de los datos de entrada, proceso denominado *clustering*. En líneas generales un *cluster* es un agrupamiento de datos parecidos.

Dentro de los procesos de aprendizaje no supervisado destaca el aprendizaje competitivo-cooperativo.

En este tipo de procesos las neuronas compiten (y cooperan) unas con otras con el fin de llevar a cabo una tarea dada. Se pretende que cuando se presente a la red cierta información de entrada, sólo una de las neuronas de salida de la red, o una por cierto grupo de neuronas, se active (alcance su valor de respuesta máximo). A esta neurona se le denomina winner-take-all unit. Las neuronas compiten por activarse, quedando finalmente una (o una por grupo) como neurona vencedora, siendo el resto anuladas es decir, forzadas a sus valores de respuesta mínimos.

La neurona ganadora es la única que actualiza sus pesos. Otra posibilidad, es dotar a la red de un cierto grado de cooperación, es decir que también las neuronas vecinas a la ganadora actualizen sus pesos, aunque en menor medida que la ganadora.

El objetivo de este aprendizaje es categorizar (clusterizar) los datos que se introducen en la red. De esta forma, las informaciones similares son clasificadas formando parte de la misma categoría, y por tanto deben activar la misma neurona de salida. La clases o categorías deben ser creadas por la propia red, puesto que se trata de un aprendizaje no supervisado, a través de las correlaciones entre los datos de entrada.

En este tipo de redes, cada neurona tiene asignado un peso total, suma de todos los pesos de las conexiones que tiene a su entrada. El aprendizaje afecta sólo a las neuronas ganadoras (activas), redistribuyendo este peso total entre sus conexiones, sustrayendo una porción a los pesos de todas las conexiones que llegan a la neurona vencedora y repartiendo esta cantidad por igual entre todas las conexiones procedentes de unidades activas. Por tanto, la variación del peso de una conexión entre una unidad i y otra j será nula si la neurona j no recibe excitación por parte de la neurona i (no vence en presencia de un estímulo por parte de i), y se modificará (se reforzará) si es excitada por dicha neurona i.

Un ejemplo de este tipo de aprendizaje son las redes SOM las cuales serán tratadas en la siguiente sección.

C. Redes Neuronales SOM

La red SOM (Mapa AutoOrganizado) es una herramienta para la visualización de datos con gran dimensionalidad de entrada, inspirada en las areas de autoorganización en el cerebro humano. En forma generica, una red SOM es un mapeo no lineal, ordenado, suave de datos de entrada de gran dimensionalidad en los elementos de una matriz de baja dimensionalidad.

C.1 Origen fisiológico del modelo SOM

Se conoce desde hace bastante tiempo que hay areas del cerebro, especialmente de la corteza cerebral, organizadas de acuerdo a diferentes modalidades sensoriales. Existen evi-

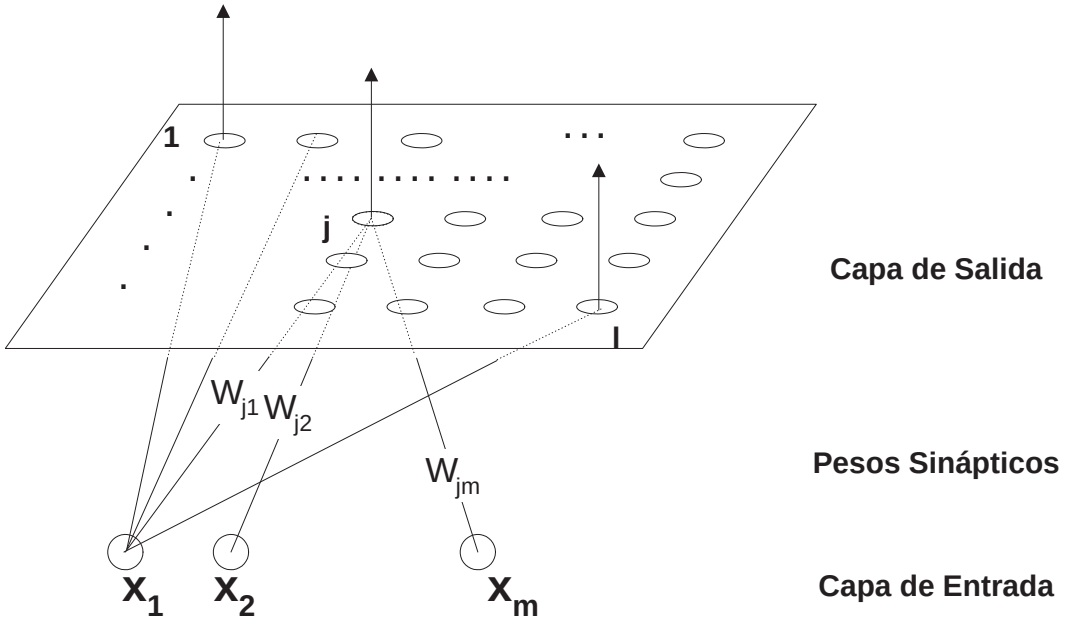

Figura 3. Red SOM de m entradas y l salidas

dencias que demuestran que en la corteza del cerebro hay neuronas que se organizan de forma que las informaciones captadas del entorno a través de los órganos sensoriales se representan internamente en forma de mapas bidimensionales. Por ejemplo, en el sistema visual se han detectado mapas del espacio visual en zonas del córtex (capa externa del cerebro). También en el sistema auditivo se detecta una organización según la frecuencia a la que cada neurona alcanza la mayor respuesta (organización tonotópica).

Aunque en gran medida esta organización neuronal está predeterminada genéticamente, existe evidencia experimental de que parte de ella se origina mediante el aprendizaje. Esto sugiere, por tanto, que el cerebro podría poseer la capacidad inherente de formar mapas topológicos de las informaciones recibidas del exterior.

A partir de estas ideas, Teuvo Kohonen presentó en 1982 [2] un sistema con un comportamiento semejante. Se trataba de un modelo de red neuronal con capacidad para formar mapas de características de manera similar a como ocurre en el cerebro. El objetivo de

Kohonen era demostrar que un estímulo externo (información de entrada) por sí solo, suponiendo una estructura propia y una descripción funcional del comportamiento de la red, era suficiente para forzar la formación de los mapas.

C.2 Arquitectura

La representación habitual de esta red suele ser la mostrada en la figura 3, donde las l neuronas de salida se disponen de forma bidimensional para representar precisamente los mapas de características.

Se trata de una red de dos capas con m neuronas de entrada y l de salida. Cada una de las m neuronas de entrada se conecta a las l de salida a través de conexiones hacia adelante (*feedforward*).

Entre las neuronas de la capa de salida, puede decirse que existen conexiones laterales de inhibición (peso negativo) implícitas, pues aunque no estén conectadas, cada una de estas neuronas va a tener cierta influencia sobre sus vecinas. El valor que se asigne a los pesos de las conexiones *feedforward* entre las capas de entrada y salida w_{ij} durante el proceso de aprendizaje de la red va a depender precisamente de esta interacción lateral.

La influencia que una neurona ejerce sobre las demás es función de la distancia entre ellas, siendo muy pequeña cuando están muy alejadas. Es frecuente que dicha influencia tenga la forma de un sombrero mejicano. Esta afirmación tiene una base biológica, ya que existen evidencias fisiológicas de interconexiones laterales de este tipo entre las neuronas del sistema nervioso central de los animales. Así, se ha podido comprobar que en determinados primates se producen interacciones laterales de tipo excitatorio entre neuronas próximas en un radio de 50 a 100 micras, de tipo inhibitorio en una corona circular de 150 a 400 micras de anchura alrededor del círculo anterior, y de tipo excitatorio muy débil, prácticamente nulo, desde ese punto hasta una distancia de varios centímetros.

C.3 Modelo básico SOM

En este modelo se trata de establecer una correspondencia entre los datos de entrada y un espacio bidimensional de salida, creando mapas topológicos de dos dimensiones, de tal forma que ante datos de entrada con características comunes se deben activar neuronas situadas en zonas próximas de la capa de salida.

Sea un espacio de entrada de dimensión m. Los patrones de entrada tienen la siguiente forma:

$$x = [x_1, x_2, ..., x_m]^T \tag{1}$$

Los pesos sinápticos de la neurona j son:

$$w = [w_{j1}, w_{j2}, ..., w_{jm}]^T, \quad j = 1, 2, ..., l \tag{2}$$

C.3.a Entrenamiento. El objetivo principal de la SOM es transformar una señal de entrada de dimensión arbitraria en un mapa discreto de dos dimensiones (también se puede mapear a otro número de dimensiones, por ejemplo a 1 dimensión).

La primera tarea que tiene el algoritmo de entrenamiento es inicializar los pesos de las neuronas. Esto se realiza asignando valores aleatorios a los mismos.

Luego de la inicialización, en cada iteración del entrenamiento, toman lugar los siguientes tres procesos:

1. Competición. Se encuentra la neurona mas próxima al patrón de entrada (Ver figura 4). Si se identifica con i(x) el índice de la neurona ganadora para el patrón x se tiene que la forma de resolver el proceso competitivo es:

$$i(x) = \min_j \parallel x - w_j \parallel, \quad j = 1, 2, ..., l \tag{3}$$

Se puede seleccionar cualquier distancia para encontrar la neurona ganadora. Por lo general se utiliza la distancia euclidea. Según cual sea la aplicación de interés la

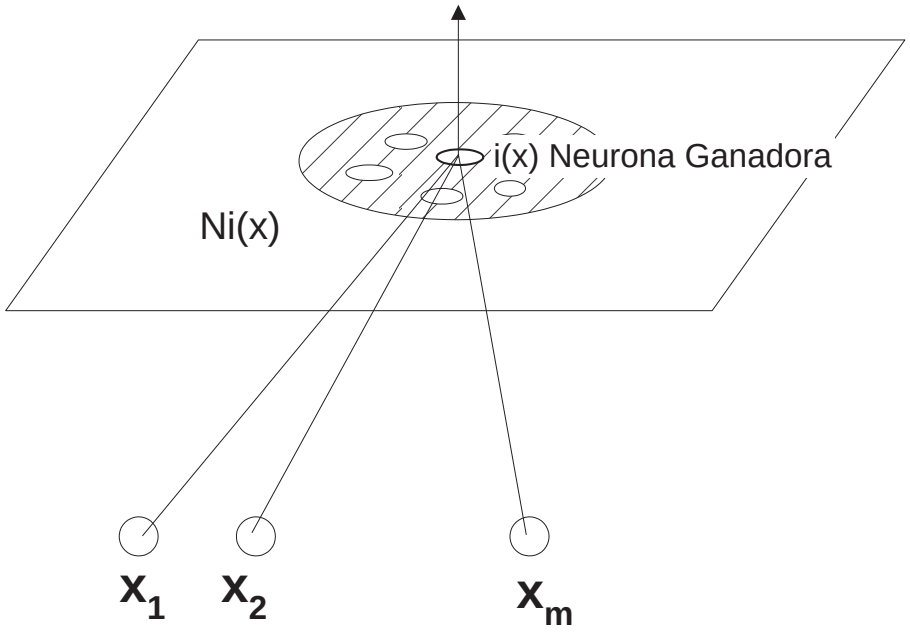

Figura 4. Proceso de competición

respuesta de la red a la entrada x puede ser el indice de la neurona ganadora i(x) o los pesos de la neurona ganadora $w_{i(x)}$

2. Cooperación. Alrededor de la neurona ganadora se define una región de neuronas que también resultarán excitadas por el patrón de entrada. Dicha región se denomina vecindad de interacción lateral $N_{i(x)}$, y esta definida por medio de la función de interacción lateral $h_{j,i(x)}$. Esta función especifica la "porción" del aprendizaje de la neurona ganadora i(x) que va a realizar la neurona j por interacción lateral. Sea $d_{j,i(x)}$ la distancia entre neuronas. Se le exigen dos condiciones a la función $h_{j,i(x)}$

   a) Ser simétrica respecto de la unidad ganadora, presentando un máximo en el centro. Esta asunción es para simplificar los cálculos y destacar el carácter de neurona ganadora del centro de la región.

   b) La amplitud de la región debe decrecer monótonamente a cero a medida que $d_{j,i(x)}$ crece. La evidencia neurobiológica justifica esta hipótesis, dado que la

interacción lateral entre neuronas es mayor entre la neurona excitada y las neuronas mas próximas, haciendose mas débil a medida que aumenta la distancia. Además esta es una condición necesaria de convergencia del algoritmo.

Hay dos tipos de funciones estándar para este cometido:

a) Función burbuja (*bubble*)

$$h_{j,i(x)} = \begin{cases} 1 & si\ d_{j,i(x)} \leq \sigma \\ 0 & si\ d_{j,i(x)} \geq \sigma \end{cases} \tag{4}$$

b) Función gaussiana

$$h_{j,i(x)} = e^{-\frac{d_{j,i(x)}^2}{2\sigma^2}} \tag{5}$$

El parámetro σ es el ancho efectivo de la vecindad de interacción lateral.

La red neuronal puede estar dispuesta en forma rectangular, es decir cada neurona va a tener 4 neuronas vecinas (en el caso general), o en forma hexagonal donde se tendrán 6 neuronas vecinas

3. Actualización de pesos. Para asegurar el comportamiento de autoorganización es necesario que los pesos sinapticos de las neuronas cambien de forma tal que los pesos actualizados luego de la iteración n $w_j^{(n+1)}$ en la vecindad de la neurona ganadora i(x) se tiendan a parecer al patron x, siendo mayor el cambio en cuanto mas cerca se este de la neurona ganadora y manteniendose invariantes los pesos del resto de las neuronas. La regla propuesta por Kohonen es:

$$w_j^{(n+1)} = \begin{cases} w_j^{(n)} + \eta h_{j,i(x)}(x - w_j) & si\ j \in N_{i(x)} \\ w_j^{(n)} & si\ j \notin N_{i(x)} \end{cases} \tag{6}$$

El parámetro η es el ritmo de aprendizaje de la red. Al recorrer el conjunto de los patrones de entrenamiento en sucesivas iteraciones se realizan dos etapas de entrenamiento:

a) Fase de autoorganización. Se realiza con un valor relativamente alto de η, del orden de 0.1, y tiene como objetivo llegar rapidamente a una mapeo inicial del espacio de entradas. Se realiza del orden de 1000 iteraciones en esta fase.

b) Fase de convergencia. Se realiza con η del orden de 0,01 y tiene por fin que la red se ajuste en forma fina a las caracteristicas del conjunto de entrada. Se realizan al menos 100 veces la cantidad de neuronas de la red.

Una vez entrenada la red se define como error de cuantificación del patrón x en la red neuronal a:

$$e(x) = \min_j \parallel x - w_j \parallel, \quad j = 1, 2, ..., l$$

Se puede medir el grado de ajuste de los datos a un conjunto de entrenamiento realizando el promedio de los errores de cuantificación para todos los patrones de entrada.

Para visualizar los clusters de datos generados por SOM existe el método de la *U-matrix*, el cual consiste en crear una matriz gráfica para la neuronas de la capa de salida, la cual tiene espacios en blanco entre ellas. Se calcula las distancias entre los vectores de pesos de las neuronas adjacentes y la misma se representa como un color gris en el espacio intermedio entre neuronas, el cual tiende a ser mas claro si las distancias son mas reducidas o mas oscuro si las distancias son grandes. Esto permite ver claramente los clusters formados, y en general esta información se guarda en formato *eps*[5].

C.4 Implementación de un clasificador basado en SOM

Hemos visto que la red SOM es un excelente medio para visualizar los clusters de un conjunto de datos de alta dimensionalidad de entrada. Sin embargo para utilizar todo este potencial en forma efectiva para el reconocimiento de patrones se debe contar con algún mecanismo que le proporcione al sistema la información de las clases a las cuales pertenecen

[5] Encapsulated PostScript

los patrones del conjunto de entrenamiento.

Es decir para identificar a que clase corresponden los clusters insinuados por el aprendizaje no supervisado, necesitamos la información de un experto que "etiquete" los patrones de entrenamiento con la clase a la que pertenecen

El mecanismo elegido para la clasificación es en este caso la votación por mayoria, la cual consiste en que partiendo del mapa realizado por SOM y la información del experto la clase de cada patrón de entrenamiento se le asigne una clase a cada neurona de salida cual sea la clase de la mayoría de los patrones de entrenamiento que tienen a esa neurona como neurona ganadora.

Una vez asignada las clases a cada una de las neuronas, se pueden clasificar otros patrones, fuera del conjunto de entrenamiento, determinando cual es la clase de la neurona mas parecida al patrón de entrada (neurona ganadora). Esta clasificación se puede mejorar si además se tiene en cuenta en forma ponderada las clases de las neuronas vecinas a la ganadora.

D. Antecedentes de implementación

Existen los siguientes paquetes software (según [1] y [3]) que implementan SOM:

1. **SOM_PAK**. Es un paquete de dominio publico programado en lenguaje *C* [4], diseñado para tareas intensivas en cálculos y de gran porte de datos. Fue desarrollado por el Laboratorio de Computación y Ciencia de la Información de la universidad de Tecnología de Helsinki. Su última versión es del año 1996. Existen versiones para plataformas **Unix** y para **Windows**. El tamaño del mapa y la dimensionalidad de los vectores son virtualmente ilimitados. Este paquete es la base de muchas de la aplicaciones profesionales de SOM.
2. **Matlab Toolbox**. La universidad de Tecnología de Finlandia también desarrollo un

toolbox de SOM para **Matlab 5** en 1996 [5]. En comparación con **SOM_PAK** es mucho mas lento y no puede procesar redes muy grandes.

3. **Viscovery SOMine**. Es una plataforma comercial [6] de data mining con opciones de preprocesamiento de los datos, implementación del algoritmo estándar, desarrollada en **Windows NT** por la empresa Eudaptics Gmbh, optimizada para aplicaciones financieras.

   Además de las funciones estándar de SOM contiene otras mas avanzadas como ser:

   - Busqueda de *clusters*
   - Información estadística de los *clusters*.
   - Evaluación de dependencia entre variables.

4. **Nenet**. Esta herramienta [7] esta focalizada en ser amigable para el usuario, con una buena interfaz gráfica, pero diseñada para problemas de pequeña escala. Corre en plataforma **Windows**. Fue desarrollada por un grupo de la universidad de Tecnología de Helsinki, y su mayor fortaleza son las técnicas de visualización implementadas.

III. Spam

A. Introducción

Describiremos la terminología a utilizar para familiarizarnos con el problema planteado. A continuación se buscará analizar el origen del fenómeno y los puntos para atacarlo eficazmente, conociendo de antemano cuales podrían ser sus características.

B. Terminología

El origen de la palabra *spam* tiene raíces estadounidenses y deriva de una marca de carne en lata originalmente llamada *Hormel's Spiced Ham*. El gran éxito del invento lo convirtió con el tiempo en una marca genérica, tan conocida que hasta el mismo fabricante le recortó el nombre, dejándolo con solo cuatro letras: *SPAM*. El popular uso de éste producto dió lugar a un sin fin de anécdotas recordandose el uso repetido de la palabra en *sketchs* cómicos y comerciales, asociandose su empleo a la repetición de un texto sin sentido o significado, en definitiva, el mismo uso del *spam* encontrado en el correo electrónico [8].

El término *spam* surge ante la necesidad de definir a aquellos mensajes de correo electrónico que llegan masivamente de manera **no solicitada y duplicativa**. Es difícil establecer una definición general, debido a que para distintos usuarios un mismo mensaje podría considerarse *spam* o no. Un sistema *antispam* efectivo debe ser capaz de adaptarse a la "definición" de *spam* de la persona que lo utilice.

Mensaje **no solicitado** significa que el receptor no dio un permiso verificable al remitente para que éste le envíe el mensaje. La visita a una página web, el llenado de un formulario o el envío de un mensaje que no solicita explícitamente recibir mensajes del tenor del mensaje en cuestión, no constituyen solicitud ni permiso para el envío de dicho mensaje.

Mensaje **duplicativo** significa que el mensaje es enviado como parte de una colección

mayor de mensajes, donde todos tienen el contenido sustancialmente idéntico. El hecho de que un mensaje incluya alguna peculiaridad del receptor (por ejemplo su nombre y apellido) no lo hace sustancialmente distinto de otros mensajes en los cuales el resto del contenido es idéntico. En este contexto no hay un límite inferior definido, por debajo del cual se pueda decir que un conjunto de mensajes duplicativos no incluyen uno o más mensajes que son *spam*. En cualquier conjunto de dos o más mensajes que tengan contenido sustancialmente idéntico, cualquier mensaje de ese conjunto que sea no solicitado por los receptores que lo recibieron, es calificado como *spam* según [9].

Existe *e-mail* no solicitado que es *e-mail* normal: ejemplos de esto incluyen requerimientos de primeros contactos, requerimientos laborales, requerimientos de ventas, etc. Existe e-mail masivo que es e-mail normal: ejemplos de esto incluyen boletines de suscripción, listas de discusión, listas de información, etc.

El principal problema, desde el punto de vista técnico, es que no se puede decir que un mensaje en particular es o no es *spam* tan sólo examinando el contenido del mensaje en sí. Para saber si es duplicativo, se debe verificar que existe al menos otro mensaje cuyo contenido es sustancialmente idéntico. Para saber si fue solicitado o no, se debe consultar al dueño de la dirección a la cual el mensaje fue enviado.

Esta es la causa de que no exista (ni pueda existir) una solución técnica automática, completa y a prueba de fallas.

Desde el punto de vista legal un mensaje electrónico es *spam* si: La identidad personal del receptor y el contexto son irrelevantes porque el mensaje es igualmente aplicable a muchos otros receptores potenciales; y no se puede verificar que el receptor haya dado un permiso deliberado, explícito, y aún revocable, para que el mismo le sea enviado [9].

Para denominar el correo electrónico normal, utilizaremos el término *ham*, bastante manejado en la bibliografía [10], mientras que a los creadores de *spam* los definiremos como *Spammers*.

Volumen de *spam*. El *spam* es un fenómeno tan antiguo como el sistema de Correo Electrónico. Desde su inicio hasta nuestra época, ha tenido un crecimiento exponencial. Segun [11] la presencia del *spam* fue la siguiente:

- 1978 - Un *spam* es enviado a 600 direcciones de *e-mail*
- 1994 - Primer *spam* enviado a 6000 *Newsgroups* alcanzando a 1.000.000 de personas
- 2005 - 30.000 millones de *spam* por día
- 2006 (Junio) - 55.000 millones de *spam* por día
- 2006 (Diciembre) - 85.000 millones de *spam* por día
- 2007 (Febrero) - 90.000 millones de *spam* por día

C. Análisis del problema

El sistema de Correo Electrónico, o *e-mail*, existe desde hace ya casi tres décadas. Los primeros sistemas consistían en protocolos de transferencia de archivos, con la convención de que la primera línea del mensaje, o sea del archivo, contendría la dirección del destinatario. Ésta carencia de estructura interna generó problemas de procesamiento en computadora. La interfaz de usuario era difícil de implementar y era poco práctico para por ejemplo insertar un mensaje dentro de otro, o envíar el mismo mensaje a listas de destinatarios. Otra dificultad era que el editor del mensaje no estaba integrado al sistema transmisor.

En 1982 se publican las propuestas de correo electrónico de ARPANET como la RFC 821 para el protocolo de transmisión y la RFC 822 para el formato del mensaje, las cuales se han convertido en estándares de facto en Internet. Por su parte, la CCITT elaboró la

recomendación X.400 mucho mas compleja y difícil de implementar. Si bien el RFC 822 no es tan bueno como se quisiera, es mas sencillo y funcional. Muchas organizaciones lo han adoptado como herramienta, por lo cual concentraremos nuestro análisis en éste tipo de sistema de correo, el mas usado en Internet.[8]

Normalmente el sistema de correo electrónico consiste en dos subsistemas: los **agentes de usuario** que permiten leer a través de un editor y enviar correo electrónico a través de una interfaz, y los **agentes de transferencia** de mensajes que son generalmentes *daemons* que operan en segundo plano y mueven correo de un sistema a otro.

Dado que se trata de una herramienta creada en principio para comunicación de usuarios de redes de computadoras, con el crecimiento de las mismas, la trazabilidad de los mensajes de correo se ha vuelto muy compleja. La verificación de un mensaje recibido es cada vez más difícil, siendo necesario determinar algunas pautas para identificar el tipo de mensaje recibido. Un criterio posible es el análisis de los contenidos.

Análisis de los contenidos. El contenido de los mensajes de *spam* y de *ham* podría analizarse mediante técnicas de clasificación de texto.[10] Éstas consisten en determinar primero los grupos temáticos de interés, y sobre éstos clasificar los mensajes.
Por ejemplo podríamos clasificar los textos en los siguientes grupos:

1. Medicina / Privado
2. Medicina / Negocios
3. Inversiones / Privado
4. Inversiones / Negocios

Utilizando algún método estadístico, cada nuevo mensaje entrante se ajustará a éstas categorías, con una etiqueta (*spam* o *ham*) y un cierto grado de confianza en la decisión:

1. Spam / 1.00

2. Spam / 0.66
3. Spam / 0.33
4. Ham / 0.66

Para el ejemplo claramente se ve que el mensaje será marcado como *spam*.

Lamentablemente, un agrupamiento manual de todo un repositorio de *mails* es poco factible, es necesario un agrupamiento automático que identifique los subgrupos.

Por lo tanto surgen dos paradigamas:

1. Identificación de los *mails* por su características técnicas
   - Encabezado
   - Tamaño del mensaje
   - Contenido de gráficos
2. Análisis de contenido del mensaje

Una aplicación *antispam* deberá tener en cuenta estos dos puntos de vista.

D. Características del spam

Basándose en ésta tecnología de correo, no preparada para abordar el problema del *spam*, los *Spammers* han logrado desarrollar programas que generan miles de millones de *spam mails* diariamente.

La actividad de los *spammers* se puede sintetizar en las siguientes etapas:[12]

1. Recabar información de direcciones de correo, ordenadolas en grupos de destino
2. Crear plataformas para envío masivo de *mails*, servidores o computadores individuales
3. Crear programas de envío masivo de *mails*
4. Ofrecer el servicio de *spam* a interesados
5. Desarrollo de textos en el asunto y cuerpo del mensaje específicos para ciertas categorías de *spam*

6. Envío del *spam*

Cada proceso generalmente es realizado independientemente de los otros.

En lo referente al punto 5 podemos decir que el *spam* pertenecerá por lo general a alguna de las siguientes categorías

- Mensajes Políticos
- Llamados a la caridad
- Ofertas financieras
- Cadenas
- *Spam* malicioso capaz de auto difundirse

Algunas observaciones sobre las modalidades de *spam* actuales.

- En un intento de camuflar el contenido de los mensajes, pueden incluir cadenas de caracteres invisibles en el texto y así interferir las soluciones de detección basadas en filtros Bayesianos.
- La inclusión de gráficos simples o dinámicos, dificulta actualmente el reconocimiento del mensaje.
- La dinámica del cambio en los textos de los mensajes, hace que las soluciones *antispam* no sean capaces de detectarlos la primera vez que llegan al cliente.

IV. Descripción general del sistema

A. *Introducción*

Con base en los conceptos manejados hasta ahora, nos proponemos diseñar un sistema que implemente la Red Neuronal en un hardware apropiado, que permita utilizar las funciones básicas del SOMPAK, como también otras funciones que implementaremos oportunamente para la aplicación anti*spam* específicamente.

Como una aproximación a lo que será el diseño definitivo, plantearemos una estructura de bloques de software que creemos necesaria para el desarrollo del Proyecto. Allí podemos identificar los principales componentes del sistema para su posterior desarrollo

B. *Descripción de bloques de software*

La figura 5 muestra un esquema de los sistemas involucrados en el desarrollo:

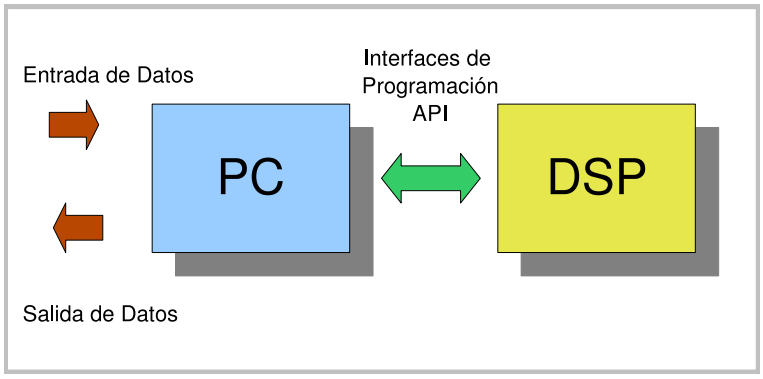

Figura 5. Diagrama de Bloques del Sistema

Internamente el sistema será tratado de la forma descripta en la figura 6.

C. *Componentes del sistema*

C.1 Bloques de Hardware

PC. Se utilizará un PC como herramienta de desarrollo, soporte de la Aplicación SOM y centro de control de procesos.

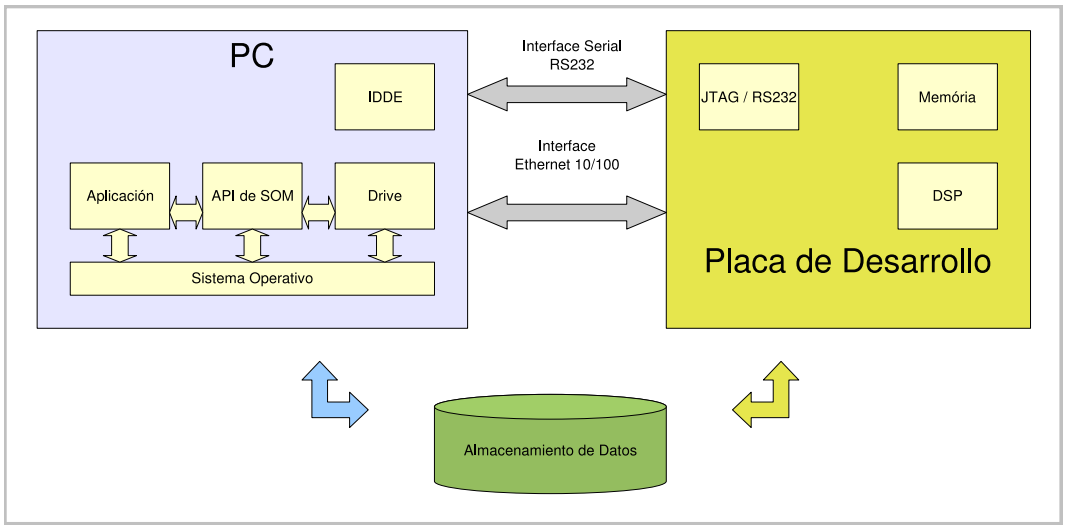

Figura 6. Diagrama de Bloques de Software del Sistema

Placa de Desarrollo DSP. La placa de desarrollo implementara el paralelismo necesario para optimizar la *performance* de las redes neuronales SOM. Ésto nos demostrará si es posible implementación del sistema en un hardware pequeño, facilmente portable, de muy bajo costo y escalable.

DSP. El procesamiento será realizado en el DSP que posee la placa,

Memoria. La placa deberá contar con memoria RAM adicional a la memoria interna del DSP que será utilizada para almacenar todos los parámetros necesarios para el entrenamiento y operación de la red neuronal.

Almaceamiento de Datos. Eventualmente, y dependiendo del volúmen de datos, el sistema deberá ser capaz de utilizar un Unidad de Almacenamiento de Datos externa.

C.2 Bloques de Software

Aplicación. Programa cuyo motor de procesamiento utiliza la red neuronal SOM.

API SOM. Conjunto de primitivas de operaciones de cálculo relacionadas con SOM

Driver de Comunicación. Interfaz programada para controlar la transferencia de datos entre el Sistema Operativo y el DSP.

Sistema Operativo. El Sistema Operativo de la Plataforma de Producción permitirá administrar los recursos de la PC de los distintos bloques de software en funcionamiento.

C.3 Plataforma de Desarrollo

La IDDE (Entorno Integrado de Desarrollo y "Debugging") que se utilizará, no dependerá necesariamente del Sistema Operativo que corra la PC de producción, la elección de la IDDE está condicionada por la plataforma de DSP elegida.

C.4 Canales de Comunicación

Puerto Serial. Canal de acceso al DSP para su programación y monitoreo

JTAG. De acuerdo a lo investigado previamente, el canal de monitoreo JTAG está implementado básicamente sobre USB o RS232.

Ethernet. Adoptaremos éste medio de comunicación, entre otras cosas, dado el tipo de aplicación que queremos implementar, o sea una aplicación de sistemas de redes de computadores.

V. Elección de Hardware

A. Introducción

Incluímos en este capitulo un informe sobre las distintas marcas de Dispositivos DSP y Otras Arquitecturas de Procesadores y su disponibilidad en el mercado.

B. Analisis de distintas alternativas ofrecidas por el mercado

A continuación presentaremos la investigación realizada en busca de determinar cuál plataforma de desarrollo utilizaremos para implemtentar el diseño. Si bien estabamos orientados a utilizar un DSP y entorno de desarrollo propietario del producto, no descartamos la posibilidad de considerar otras arquitecturas para sistemas embebidos, y otro concepto para el entorno de desarrollo como lo es el GNU. De nuestra investigación surgirá el dispositivo con el que trabajaremos en el proyecto en adelante, por lo tanto consideramos esta etapa medular para el éxito del proyecto.

B.1 Arquitecturas DSP

Basándonos en la publicación de la revista electrónica EDN [13], obtuvimos un listado actualizado del año 2005 y anteriores años, de las distintas marcas de fabricantes de procesadores DSP. Encontramos coincidentemente algunos dispositivos ya vistos anteriormente como es la línea BlackFin de Analog Devices o RC Module.

Listado de Marcas

- 3DSP
- ADS (Analog Devices)
- ALTERA
- ATMEL
- Equator

- LSI
- Morpho
- Motorola (Freescale)
- Philips
- RC Module
- STMicroelectronics
- Texas

Los criterios de selección implican considerar :

- Velocidad del procesador
- Precisión
- Interfaces disponibles en placa de desarrollo
- Entorno de Desarrollo de fácil obtención
- Precios

En general todas las marcas poseen al menos un procesador con características adecuadas al proyecto, pero en la mayoría de los casos la mayor limitante fueron los precios de las herramientas de desarrollo (Placas, Software). Como fue el caso de ALTERA, MORPHO, Texas, Philips, Motorola. Por otra parte, otras marcas si bien ofrecen dispositivos con gran capacidad de procesamiento, las placas de desarrollo no poseen interfaces adecuadas a nuestros requerimientos del proyecto, es decir, bus PCI , USB 2.0, Ethernet u otros. Tal fue el caso de los productos de LSI, Equator y algunos modelos de Analog Devices como Tiger SHARC. Otro inconveniente encontrado fue la indisponibilidad de soporte técnico o inclusive comercial en algunas de las marcas, dado que los productos se encontraban dirigidos a la industria y no al estudio o desarrollo por parte de estudiantes. No pudimos obtener más datos de por ejemplo 3DSP, HyperStone y Philips. En el caso de ATMEL, el DSP y la placa de desarrollo cumplían los requisitos iniciales pero luego de consultar a la empresa se

obtuvo la información de que la placa de desarrollo estaba fuera de producción. Finalmente concentramos nuestra atención en dos marcas, en primer lugar la línea BlackFin de ADS y por otra parte el procesador NM6403 de la empresa RC Module. Ambas marcas ofrecen procesadores con capacidades para nuestro proyecto pero no sin inconvenientes.

1. *RC Module*

Comenzaremos con el primero que fue descartado, es decir el dispositivo NeuroMatrix NM6403 de RC Module. Esta empresa con sede en Rusia, ha desarrollado un procesador RISC-DSP con arquitectura matricial con flexibilidad de elección de precisión y performance, ancho de palabra de 64 bits y hasta 50 MMACS (Million Multiply Accumulate Cycles per Second) para precisión de 32 bits. Dada su flexibilidad puede ser utilizada como sistema DSP, en el campo de procesamiento de señales, en comunicaciones inalámbricas, procesamiento de imagen o aceleración en redes neuronales. La empresa ofrece una placa de desarrollo, MC 431 con interfaz PCI, dos bancos de memoria SRAM de 2MB accesibles para lectura o escritura desde el Bus PCI y dos puertos de comunicación. El costo de la misma es de USD 750 menos un descuento de 30 por ciento para estudiantes, pero de todas formas excede el presupuesto.

Además, la empresa ofrece un SDK (Software Development Kit) que contiene:

- Compilador C++;
- Assembler;
- Linker;
- Librerías;
- C- Debugger;
- Drivers para Windows;
- Librerías C run-time códigos fuentes;

El mismo en su versión completa tiene un costo demasiado elevado y la versión de pruebas

no contiene todos los componentes descriptos. Dada su procedencia no pudimos encontrar otros compiladores y ambientes alternativos para ser utilizados con dicha placa de desarrollo.

2. *Blackfin Analog Devices ADSP-BF5xx*

Es una familia de procesadores dual-MAC y punto fijo de 16 bits, de propósito general. Utiliza un set de instrucciones mixto de 16/32 bits, o combinado formando un sistema de instrucciones VLIW. Las primeras generaciones obtienen una velocidad de hasta 350 MHz, como es el caso de BF535, la segunda generación, BF531/32/33 y BF561 comienzan a operar a partir de 750 MHz.

La familia ADSP BF5xx en general se utiliza para procesamiento de datos en valores de 16 bits y 8 bits para video. Muchas instrucciones soportan datos a 32 bits y algunas hasta 40 bits, por ejemplo soporta multiplicaciones de 32 x 32 a 32 bits, consumiendo 5 ciclos de reloj.

La memoria esta organizada en un espacio unificado de 32 bits, además de la memoria on-chip L1 en bancos separados de datos e instrucciones. La forma de organización de dicha memoria es el principal diferenciador de las distintas arquitecturas BF5xx, que manteniendo su base en la arquitectura Harvard, permite configurar parte de esa memoria en cache. Posee además 256 Kb de memoria L2 para programa y datos. Las características del procesador permiten implementar *pipeline* de hasta 8 etapas, dependiendo del tipo de instrucciones.

En lo que respecta a periféricos, la familia ADSP BF5xx ofrece gran variedad de los mismos incluyendo controladores DMA, I/O pins de propósito general, puertos UART, 2 Seriales. En particular el BF535 posee interfaces USB y PCI, y el BF537 posee interfaces USB y Ethernet.

Las herramientas de desarrollo disponibles son variadas, dónde se destaca la presencia en todas de compiladores C/C++, librerías , assembler y linkeador. De acuerdo al grado de

Fabricante DSP	Modelo DSP	Placa Desarrollo	SDK	Interfaces Disponibles	Velocidad (MHz)	Precisión *bits*	Costos USD
RC Module	NM6403	MC431	NeuroMatrix	PCI	100 MHz	64	525
Analog Devices	BF535	HAWK-35	VisualDSP	PCI	300 MHz	16-32	1,495
	BF535	PDE 535	VisualDSP	PCI, USB	300 MHz	16-32	895
	BF535	EZ-KIT	VisualDSP	PCI	350 MHz	16-32	295
	BF533	EZ-KIT	VisualDSP	Eth/100	600 MHz	16-32	680
	BF533	STAMP	uClinux	Eth/100	500 MHz	16-32	169
	BF537	STAMP	uClinux	Eth/100	500 MHz	16-32	225
ST Micro	ST 140						
Texas	TMS320	LM1000	Code Composer	PCI	400 MHz	32	995

TABLE I
SELECCIÓN DE DSP

Emulación, simulación y debugg que busquemos tenemos distintas posibilidades de las cuales se destacan 3:

- VisualDSP++ 4.0 de Analog Devices
- MULTI(R) de Green Hills Software
- Open Source GCC y uClinux

En lo referente a placas de desarrollo se tiene una gran variedad de las mismas existiendo dos grandes familias: EZ-LITE, proporcionada por el fabricante del DSP y STAMP para uso de software libre.

Los precios encontrados en el mercado son compatibles con el presupuesto del proyecto en algunos casos, pero para placas que no ofrecen las interfaces deseadas, y en otros casos con todas las interfaces (PCI, USB 2.0, Ethernet) pero a costos más elevados.

A continuación se detalla en la tabla I las distintas variantes de procesadores, placas de desarrollo con sus respectivas interfaces y los costos aproximados.

B.2 Otras Arquitecturas de Procesadores para sistemas embebidos

Realizamos una búsqueda exhaustiva de arquitecturas y procesadores ofrecidos por las distintas marcas existentes. Nos basamos nuevamente en la publicación de EDN para obtener un listado actualizado de Microprocesadores y Microcontroladores de 32 bits para el año 2005.

Nuevamente los criterios de selección del extenso listado de procesadores, mayor aún que el de DSP fue considerar:

- Velocidad del procesador
- Precisión
- Interfaces disponibles en placa de desarrollo
- Entorno de Desarrollo de fácil obtención
- Precios

1. *Marcas existentes al 2005* Con la expectativa de tener una mayor oferta de productos realizamos la búsqueda para las siguientes marcas:

- AMCC
- AMD
- ARM
- Cradle
- FreeScale
- Hyperstone
- IBM
- IDT
- Intel
- MIPS
- NetSilicon

- QuickLogic
- Renesas
- ST
- Stretch

2. *Arquitecturas*

De la gran diversidad de arquitecturas existentes podemos resaltar que en la mayoría de los procesadores se presentan algunas de las siguientes arquitecturas o actualizaciones de ellas:

- ARM
- PowerPC
- MIPS
- ColdFire
- SuperH
- ARM

Advanced RISC Machine

Es como su nombre lo indica un procesador con arquitectura RISC, desarrollado inicialmente por la empresa Arcon llegando esta a producir un procesador con bus de datos de 32 bits, espacio de direcciones de 26 bits y 16 registros de 32 bits. El gran propósito de este diseño fue de optimizar el uso de transistores, alrededor de 30,000 comparado con los 70,000 del 68K de Motorola, mediante un reducido microcódigo, tan solo 1/4 de dicho procesador. El diseño fue adoptado por la empresa Apple posibilitando un mayor desarrollo del mismo hasta nuestros días donde distintas marcas han licenciado esta arquitectura manteniendo el bajo número de procesadores posibilitando de esta forma un bajo consumo. Especialmente utilizado en discos duros, telefonía móvil, routers, calculadoras, juegos. Hoy día puede ser encontrado en aproximadamente 75 por ciento de los sistemas embebidos de 32 bit.

PowerPC

Se trata de una arquitectura RISC creada en la década del 90 por Apple, IBM y Motorola basado en el microporcesador POWER uno de las primeras implementaciones RISC a gran escala como respuesta a los diseños MIPS y SPARC. Existen versiones en 32 y 64 bits, con soporte a operaciones *big endian* y *little endian* en tiempo de ejecución. Simple y doble precisión para instruciones en punto flotante. Inicialmente utilizado para sistema operativo OS/2 de IBM y adaptado a MAC OS de Apple. Tiene gran desarrollo en esta área, como CPU de sistemas de computos tales como RS/6000 y AS/400 por parte de IBM (actualmente los fabrica AMCC). Por su parte Motorola se basó en esta arquitectura para desarrollar la línea de procesadores PowerQUICC como lo son los MPC8xx y MPC5xx utilizados en el área de comunicaciones fijas y wireless. Pueden ser encontrados en Computadores IBM, Apple, Amiga, routers CISCO, juegos como PlayStation 3, Xbox y Nintendo productos a lanzados en el año 2006. Es uno de los favoritos en sistemas embebidos de 32 bits, obteniendo en un chip además del CPU, memoria cache, señal de reloj, temporizadores, memoria SDRAM, periféricos, y buses PCI ,PCI-X, ROM/Flash.

MIPS

Microprocessor without **I**nterlocked **P**ipeline **S**tages, es un microprocesador RISC basado en el desarrollo de la Universidad de Standford en los años 80, donde la idea es aumentar la performance mejorando la técnica de *pipeline* muy difícil de implementar, ya que requiere el uso de interbloqueos entre instrucciones que demandan demasiado tiempo de CPU. El diseño MIPS resolvió este problema haciendo que todas las instrucciones sean ejecutadas en 1 ciclo de reloj. Existen versiones en 32 y 64 bits conteniendo unidades FPU (Floating Point Unit) y varios dispositivos en interfaces de entrada / salida. Ampliamente utilizado en Routers, modems, ADSL modems, SmartCards, Impresoras lasers, Robots y Juegos (PS2).

ColdFire

Es un desarrollo basado en la arquitectura 68K de Motorola (en la actualidad FreeScale), de bajo costo, escalable y con arquitectura RISC. De bajo consumo de memoria gracias a la implementación on-chip de la misma permitiendo empaquetar código en memoria. Especialmente dirigido a desarrollo de sistemas embebidos de 32 bits, ofrece la posibilidad de parametrización de componentes y configuración de memoria de acuerdo a la aplicación.

SuperH

Arquitectura RISC basada en el diseño del H8 de Hitachi, actualmente producida por Hitachi y ST Microelectronics. El diseño SuperH actual agrega el manejo de instrucciones SIMD (Single Instruction, Multiple Data) . Los diseños anteriores son producidos por la empresa Renesas. Existen versiones en 32 y 64 bits con velocidad a partir de 200Mhz utilizados en la industria de entretenimiento.

Resultados de la Investigación

Luego de una primer clasificación con los criterios anteriormente vistos se llegó a un listado menor de posibles procesadores . Finalmente debimos buscar las herramientas de desarrollo disponibles y los costos de las mismas en la tabla II

B.3 Criterios de selección

Dado que tenemos un panorama completo de lo que ofrece el mercado a la escala que pensamos trabajar, es decir, la adquisición de una placa de desarrollo y posiblemente el software, tenemos que considerar nuevamente:

- Velocidad del procesador
- Precisión
- Interfaces disponibles en placa de desarrollo
- Entorno de Desarrollo de fácil obtención

Fabricante Procesador	Modelo / Arquitec.	Placa de Desarrollo	SDK	Interfaces Placa	Velocidad (MHz)	Precisión *bits*	Costos USD
AMCC	405GP PowerPC	405GP Embedded Planet	405GPC	PCI Eth/100	333 MHz	32	1195
AMD	Alchemy Au1000 MIPS32	EP1000 Embedded Planet	Linux OS	USB 1.0 Eth/100 RS 232	500 MHz	32	495
IBM	440GP PowerPC	PPC-440 Artesyn		PCI Eth/100	400 MHz	32	500
Freescale	MPC5200 PowerPC PowerQuicc	EP5200 Embedded Planet	Linux OS	USB Eth/100	400 MHz	32	795
	MCF5484 ColdFire	M5484LITE LOGICPD	Linux BSP Open Source	USB Eth/100	200 MHz	32	350
	MPC8250 PowerQuicc II	Rattler PCI Analogue Micro	Linux eCos GNU Toolset	USB Eth/100	266 MHz	32	595
	MPC8540 PowerQuicc III	CWH-PPC-8540N-VX	Code Warrior Linux	GigaEth PCI	667 MHz	32	1095
	MCF5282 ColdFire	WildFire Steroid Micros	GNU C/C++ Eclipse	Eth/100 Blue Tooth	64 MHz	32	399
	MCF5282 ColdFire	UCdimm Arcturus	GNU Tools Eclipse	Ethernet 10/100	66 MHz	32	215

TABLE II
FABRICANTES Y MODELOS DE MICROPROCESADORES

- Precios

C. Plataforma adoptada

Examinando los datos anteriores llegamos a la conclusión que la mejor relación entre prestaciones de la placa, respaldo y precio se obtiene con los procesadores Blackfin de Analog Devices. A su vez mirando en Internet encontramos que existe una comunidad grande de desarrolladores *Blackfin* en *uClinux*, siendo sencillo compartir experiencia con quienes estuvieran desarrollando en condiciones similares a las nuestra [14]. Dicha comunidad esta respaldada por un equipo de desarrolladores de Analog Devices en continua actividad. Por

Fabricante DSP	Modelo DSP	Placa Desarrollo	SDK	Interfaces Disponibles	Velocidad (MHz)	Precisión *bits*	Costo USD
Analog Devices	BF537	STAMP Board	uClinux GNU	Ethernet 10/100	600 MHz	32	165

TABLE III
PLATAFORMA ADOPTADA

último termino de cerrar nuestra decisión el hecho de que el paquete de placa de desarrollo y herramientas estaba en el orden de los USD 200.

VI. Adaptación del sistema al hardware disponible

A. *Introducción*

En esta sección se realizará una descripción de las capacidades del hardware dsp utilizado, el entorno de desarrollo, los desafios que presenta la adaptación de la red SOM, una descripción del paquete SOMPAK y los problemas encontrados

B. *Características del hardware disponible*

El hardware sobre el cual se va a implementar la parte SOM de la solución antispam es la placa de desarrollo **bf537 stamp**, ver figura 7. Esta placa forma parte del proyecto *open source* **Blackfin/uClinux** [14].

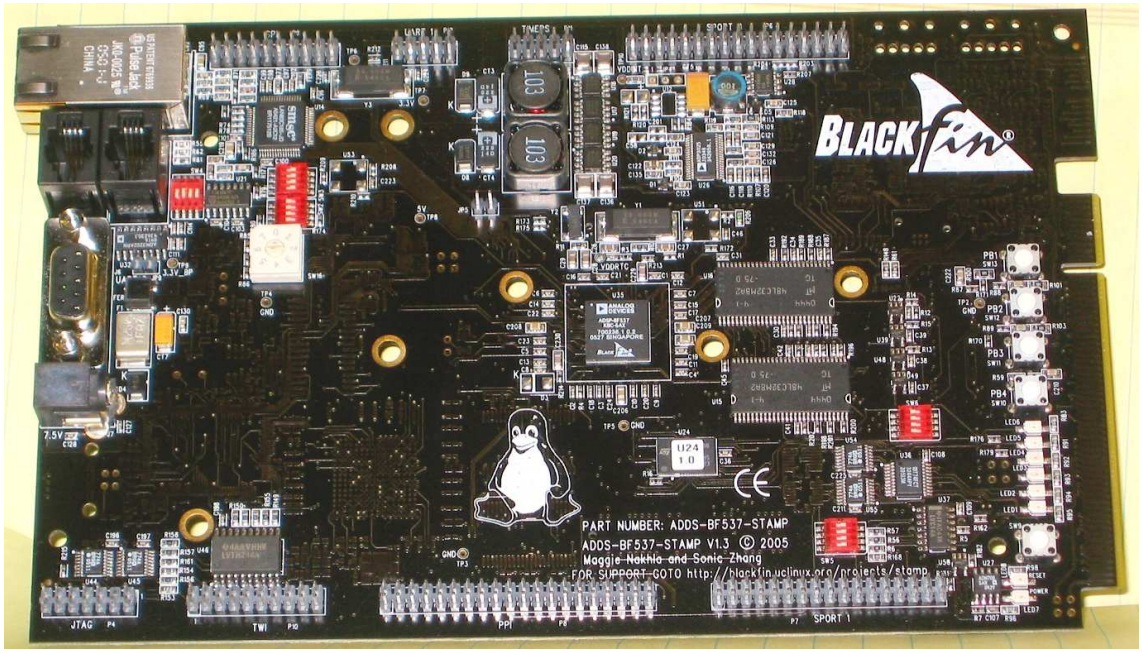

Figura 7. Placa *bf537 stamp*

Las características de la placa son:

- Procesador **Bf537 Blackfin** a 500MHz. Ver apéndice A
- Reloj de sistema a 125MHz.

- Memoria SDRAM de 64MB.
- Memoria flash de 4MB.
- Puerto Ethernet 10/100 Mbps conectado via DMA al modulo onchip MAC.
- Interfaz CAN doble.
- Conector JTAG de 14 pines.
- 6 LEDs de proposito general.
- 4 pulsadores de proposito general.
- Puertos de expansión

La comunicación a la PC se puede realizar de las siguientes maneras basicas:

- Ethernet
- Serial
- JTAG

Una característica importante de esta placa es que están disponibles en [14] los esquemáticos para construirla.

La placa elegida esta intimamente relacionada con un conjunto de herramientas de desarrollo basadas en la *gnu toolchain* (*gcc*, *gdb*, *gas*, *gprof*, *gnu binutils*, etc.) el cual permite:

- Desarrollar software en c/c++/*assembler* de **Blackfin**.
- *Debugging* avanzado de aplicaciones
- *Profiling* de aplicaciones
- Acceder y modificar la memoria de programa.
- Acceder y modificar los registros internos del procesador.

La placa utiliza el sistema operativo *uClinux*, el cual esta basado en el *kernel* de linux 2.6.x adaptado a diversas arquitecturas de sistemas embebidos, entre ellas el **Blackfin**. Esto presenta la ventaja de que el sistema operativo ya tiene solucionada los drivers de los perifericos de la placa y la interacción entre procesos (multitarea preemptiva, scheduling,

priorización, etc.).

Como contrapartida hay un consumo de recursos asociados al *kernel* y el tiempo de booteo se ve incrementado.

Figura 8. *uClinux* para *Blackfin*

C. Entorno de desarrollo

El modelo de desarrollo en sistemas embebidos basados en *uClinux* es el de la compilación cruzada, el cual consiste en crear el codigo ejecutable para una plataforma distinta a la que corre el compilador. Se denomina sistema *host* a la plataforma en la cual corre el compilador y sistema *target* a la plataforma para la que se genera el código ejecutable. Como plataforma *host* se ha utilizado un PC con **Linux OpenSuse 10.2**.

Para desarrollar se cuenta con las siguientes herramientas:

1. *gnu toolchain*. Se empezó trabajando con la versión 2005rc2 y al cabo de tres actualizaciones se esta trabajando con la versión 2006rc2. Está compuesto por:
   a) *bfin-uclinux-gcc*. Es el compilador de C basado en gcc 4.1.1. Permite trabajar tanto en C como en assembler de **Blackfin**.
   b) *bfin-uclinux-gdb*. *Debugger* estándar basado en *gdb* 6.5. Se basa en la técnica de

debuggig remoto, la cual consiste en que gran parte del proceso de *debugging*, incluyendo el manejo de la tabla de simbolos, sea realizado por el sistema *host*. El sistema target debe tener un elemento de software, en este caso el programa *gdb*server, que se comunique con el *gdb* externo. En el caso de la placa utilizada la comunicación de ambos elementos se puede realizar por Ethernet o por puerto serie, obtandose por Ethernet.

c) *gnu binutils*. Se cuenta con la versión 2.17. Entre otras, incluye las siguienes herramientas:

- *gas*. Ensamblador para **Blackfin**, integrado plenamente al compilador de C, de forma tal que se pueden ejecutar rutinas de C desde *assembler* y viceversa.
- *linker*.
- *Elf2ftl*. Convierte objetos *Elf* en objetos *FlatElf* (posteriormente se verá el significado de estos terminos).
- *object explorer*. Visualiza el contenido de los objetos

2. *uClibc*. La última versión utilizada es la 0.9.92. Es la libreria estándar de C para sistemas embebidos. Es mucho mas liviana que la *glibc*, y sus características pueden ser habilitadas selectivamente de acuerdo a los requerimientos del problema. Corre en procesadores sin MMU. Entre otras cosas define las llamadas del sistema como: *open*, *read*, *write*, *ioctl*, *close*. En cada aplicación se debe configurar y recompilar.
3. *uClinux*. Es el sistema operativo de la placa. Esta basado en el Kernel de Linux, en concreto la última versión manejada está basada en el *kernel* 2.6. Se deben instalar los fuentes en la plataforma host, ya que es parte del proceso de desarrollo obtener una imagen del sistema operativo ajustada a las necesidades de la aplicación a desarrollar.

Esto se logra recompilando el *kernel* con la configuración adecuada a la aplicación.

La imagen del *kernel* compilado determina [15], entre otros, los siguientes aspectos:

- Estructura de directorios
- Aplicaciones contenidas
- Protocolos habilitados

4. *uboot*. Es la aplicación universal de booteo que corre la plataforma target. Permite las siguientes formas de booteo:
   - *FlashRam*. La imagen del sistema operativo proviene de la memoria RAM de la placa.
   - *tftp*. Se busca la imagen en red, asignando una dirección IP fija al sistema target, el cual carga la imagen del *kernel* a partir del protocolo de transferencia trivial de archivos *tftp*.
   - *dhcp*. Al bootear el sistema target se comunica con un servidor *dhcp*, el cual le asigna una IP dinámca temporal y le transfiere la imagen del sistema operativo.
   - *Serial boot*. Por medio del protocolo zmodem se descarga la imagen de booteo a través del enlace serial.

Además se cuenta con las siguientes herramientas que hacen mas amigable el desarrollo:

- *Eclipse IDE*. *Eclipse* [16] es un entorno de desarrollo en Java. Con él se pueden realizar las siguientes tares de *uClinux*:
  - Realizar la compilación de *kernel*
  - Desarrrollar en C/C++ con ayuda de un *plugin* [17].
  - *Debuggear* con *Eclipse* como frontend de *gdb*. Esto también se logra instalando el *plugin* [17]

  En todos los casos es necesario configurar las herramientas de la gnu toolchain de

Blackfin como herramientas externas.

- *Data Display Debugger*. *DDD* [18] es un frontend gráfico para debuggers de línea de comando, en particular los derivados de *gdb*. La vista grafica facilita mucho el desarrollo, haciendo legible la complicada salida de *gdb*. Como contrapartida, presenta algunas inestabilidades extra al *gdb* no resueltas todavía.

Para comunicarse entre *host* y *target* se utilizarón las siguientes técnicas:

1. Terminales. Fueron empleados los dos tipos de terminales indistintamente:
   - *Comunicación serial*. Se utiliza el programa *minicom* configurado para 57600 *baudios*, 8 bits de datos, 1 bit de parada y sin paridad.
   - *Comunicación ethernet*. Se utiliza el protocolo *Telnet*.
2. Transferencias de archivos. El modo preferido ha sido pasar los archivos utilizando *ftp* por el enlace *Ethernet*. Para ello se configuró un servidor *ftp* en la placa. Es también posible trnsferir archivos por el enlace serial por protocolo zmodem, pero esto se descartó debido a la versatilidad del ftp.

Al utilizar la placa se constató que no se podían utilizar bien los leds y pulsadores de proposito general de la placa. Como aplicación de prueba del entorno de desarrollo se implementó el conocido juego del **simon** [19]. Para ello fue necesario realizar ajustes a los driver linux de los leds, dado que no funcionaban en forma correcta. Los ajustes realizados fueron:

- Corrección de la tabla de definición de dispositivos.
- Correcciones a la llamada *ioctl* para el driver *pflags*.

D. Ajustes realizados al kernel

Para la aplicación a desarrollar hay que construir una imagen del *kernel* de *uClinux* que se adapte a la aplicación. Entre otras cosas hay que habilitar en *uClinux* las siguientes

caracteristicas:

- Soporte de *NFS* versión 4.
- Soporte de *elf/flat*.
- Soporte de *profiler*.
- Soporte de *gdb*.

Previo a la compilación del *uClinux* es necesario configurar la libreria general de *C* para sistemas embebidos *uClibc*. Hay que habilitar lo siguiente:

- Soporte completo de *Remote Procedure Call*. Se debe habilitar para permitir el debugging remoto.
- Soporte de *profiling*.

En el uboot ha sido necesario configurar:

- Booteo a partir de una imagen de red por *tftp*.

E. Paquete SOM_PAK

Una implementación robusta y escalable es el paquete *SOM_PAK* [4] de la universidad de Helsinki. Dada su difusión y confiabilidad se tomará como base para la implementación de la red *SOM* en el *DSP* **Blackfin**. El paquete *SOM_PAK* contiene los siguientes programas

- **randinit**. Inicializa los pesos sinápticos de la red con valores aleatorios.
- **vsom**. Entrena la red según el algoritmo *SOM*. El ritmo de entrenamiento decrece linealmente con la cantidad de iteraciones. También, a medida que se van realizando cada iteración de entrenamiento, el radio de la vecindad de interacción lateral decrece linealmente.
- **qerror**. Tomando la red *SOM* entrenada y el conjunto de patrones, este archivo genera el error de cuantización para el conjunto de patrones, es decir para cada patrón se retorna la minima distancia existente entre el patron y una neurona de la red.

- **vcal**. A este programa se le proporciona el mapa *SOM* entrenado y el conjunto de patrones de entrenamiento con etiquetas que indican a cual clase se parece mas cada neurona. Esto se logra por medio de la votación por mayoría, la cual consiste en determinar para todo el conjunto de patrones de entrada la neurona ganadora, contabilizar el número de veces que una neurona es ganadora para cada clase, y de asignar a cada neurona la etiqueta correspondiente a la clase que tuvo mas "votos". En caso de que una neurona no tenga votos no se le asigna etiqueta.
- **umat**. A partir del archivo de entrenamiento de la red SOM etiquetado se construye un archivo *eps*[6] que indica la cercania entre la neuronas (codificado en niveles de grises, cuanto mas claro mas cercanas entre sí). En dicho mapa también están reflejadas las etiquetas de las neuronas.
- **visual**. Este programa devuleve para cada patrón de entrada que se le suministre cual es la coordenada de la neurona ganadora, el error de cuantización asociado y la etiqueta de la neurona elegida.

F. Adaptación de la aplicación a la placa

Un primer inconveniente fue lograr la compilación de *SOM_PAK* en la nueva arquitéctura. Para ello fue necesario realizar diversos cambios en el Makefile del paquete, ya que varias opciones de compilación estándar en gcc no tienen un equivalente exacto en *bfin-uclinux-gcc*, siendo además dependiente del modelo runtime que utilice el gcc, como veremos mas abajo. Solucionados estos problemas el paquete compiló en **Blackfin** de forma aceptable.

Otra modificación importante realizada fue pasar a un único ejecutable som, que tiene como funciones los subprogramas del *SOM_PAK*, y al que posteriormente se le agregarán los programas de comunicaciones con la *PC*.

[6] Encapsulated PostScript

Analizando diversas técnicas para acelerar la performance se decidió probar con la implementación de paralelismo vía software utilizando la librería *pthread* [20]. *Pthread* es el estándar de POSIX 1003.1 de 1995 para implementación de *threads* en *Unix*. Está soportada en **Blackfin** desde principios de 2006 (versión 2006 rc1 de la *gnu toolchain*). Un *thread* es un proceso "liviano" que puede ser corrido con otros threads en paralelo compartiendo recursos. Es una forma de implementar paralelismo por software.

Dichas técnicas fueron probadas en un problema particular, el perceptrón de capa simple [21]. Consiste en una red *feedforward* con dos capas, con una única neurona en la capa de entrada. Para probar la red se utilizó un problema de clasificación simple con clases obtenidas de la base de datos de problemas de reconocimiento de patrones de la Universidad de Califormia [22].

En las pruebas realizadas no se observó un incremento notable de performance. Considerando que la librería pthread genera también un overhead de performance para su funcionamiento, que posiblemente no se encontrará madura la implementación de la librería en **Blackfin** y por recomendación del tutor, se decidió pasar a otro enfoque para la implementación del SOM que utilizará mas las capacidades nativas del **Blackfin**.

La estrategía a seguir es identificar las partes del código del *SOM_PAK* que son mas intensivas en operaciones. Para ello se utilizará un textitprofiler, el cual es una herramienta analisis de performance de software. Consiste en una utilidad que supervisa el comportamiento de un programa a medida que el mismo va siendo ejecutado, relacionando eventos con las partes del codigo que se van recorriendo. Existen dos tipos de *profiler*:

- Basado en eventos. Cierto tipo de eventos, por ejemplo llamadas a funciones, disparan la acción del *profiler*. En esta categoría esta *JVM*, *profiler* de *Java*.
- Estadístico. El *profiler* maneja contadores del sistema, y los utiliza para muestrear

periodicamente el estado del procesador mediante interrupciones del sistema operativo. Como ejemplo tenemos en Linux el *profiler* de *gnu gprof* y *Oprofile* [23], el cual es un *profiler gpl* basado en el *kernel* que agrega bajo *overhead* al programa.

En la arquitectura **Blackfin** se utilizaba *gprof*, el cual es parte de las *gnu binutils* e interactua de forma cercana con *gcc*. Ahora bien las últimas versiones de gcc para **Blackfin** han dejado de dar soporte a las opciones de *gprof*, cambiando al *Oprofile*. Es necesario recompilar el *kernel* de *uClinux* para activar el soporte al *profiler*.

En las pruebas realizadas, con respecto al problema modelo, se vió que el soporte del *Oprofile* es inestable, siendo inmaduro todavia en esta versión de *uClinux*. Como solución alternativa se corrió el problema estándar de *SOM* en una *PC* con *Linux*, y utilizando *Oprofile* se llegó a la conclusión que el mayor tiempo de ejecución sucede en la rutina que calcula la distancia entre patrones.

En general las capacidades especiales de los DSP no están bien modeladas por los compiladores de *C*, particularmente en lo referente a aritmética punto fijo. Es de destacar que la versión de gcc para **Blackfin** no contiene soporte de aritmética de punto fijo. Actualmente se está desarrollando un añadido al gcc para que soporte aritmética de punto fijo en varias arquitecturas, aunque de momento no en **Blackfin**.

Las rutinas de calculo de la red neuronal son intensivas en operaciones de punto flotante. Dichas operaciones se implementan en el paquete *SOM_PAK* mediante las rutinas por defecto del entorno runtime del *gcc*. Dichas rutinas son genericas y no están optimizadas para la arquitectura **Blackfin**.

La estrategía a seguir para adaptar el software a la arquitectura va a ser escribir rutinas en *assembler* de manejo de punto flotante relacionadas con la distancia de forma tal que se adapten mejor a la arquitectura, las cuales serán llamadas por el resto de las rutinas en *C*

del *SOM_PAK*.

Por tanto es necesario estudiar:

Interfaz Binaria de Aplicación en *uClinux*

Consiste en la interfaz de bajo nivel entre un programa de aplicación y el sistema operativo y entre componentes de una aplicación, detalla entre otras cosas como se implementan los tipos de datos, las convenciones a seguir para llamar las rutinas en *C/C++/Assembler* (como pasar argumentos de entrada, como devolver los argumentos de salida, etc.).

Todo esto depende del modelo de *run time* elegido que este utilizando el *uClinux*. Hay tres posibles modelos:

1. *ELF. Ejecutable and Linkable Format* Es el formato de estándar de *Linux* para archivos ejecutables. Se utiliza para aplicaciones que corren sin el sistema operativo.
2. *Flat ELF*. Es el formato *ELF* adaptado a los sistemas embebidos, con énfasis en reducir los requerimientos de memoria, y con un entorno de *run time* mas eficiente que el correspondiente a *ELF*.
3. *FDPIC ELF*. Es un formato de *Linux* que da soporte a la combinación de codigo independiente de posición y objetos compartidos dinámicos. Ha sido incorporado recientemente a *uClinux* de **Blackfin**.

Se optara por *Flat ELF*, dado que es el soporte con mayor desarrollo en el *uClinux* de **Blackfin**. Es necesario configurar dicho soporte en el *kernel*.

Para este tipo de modelo de *run time* el formato del tipo de datos float en gcc es:

Tipo	Nº bits	Representación	Representación en Registro/Memoria
float	32	32 bit IEEE de precisión simple	seeeeeeeemmmmmmmmmmmmmmmmmmmmmm

TABLE IV
FORMATO DE PUNTO FLOTANTE PRECISIÓN SIMPLE EN *Flat ELF*

Las llamadas de rutinas se deben realizar de la siguiente forma:

1. Pasaje de argumentos. La lista de argumentos se pasa en el stack, utilizando una palabra para cada uno de ellos. Los tres primeros argumentos se pasan en por medio de los registros de datos R0, R1 y R2. Si un argumento es mas grande que una palabra se ocupa la cantidad de palabras que corresponda. En caso de que el argumento sea una estructura se pasa de la misma manera un puntero a la estructura.
2. Valores de retorno. Se retorna en R0, si el valor de retorno es menor o igual de 32 bits. Entre 32 y 64 bits se retorna en el par R0-R1. En caso de retornar objetos mas grandes se devuelve un puntero al objeto en P0.
3. Utilización de registros. Se deben salvar en el stack los registros R3-R7 y P3-P5, si la función llamada hace uso de los mismos. Los registros de stack SP y FP deben apuntar a areas válidas del stack. Los registros L0-L3 se deben mantener en 0. El resto de los registros puede modificar sus valores como considere necesaria la aplicación llamada.

Aritmética del Blackfin

El DSP **Blackfin** es de punto de fijo y trabaja con palabras de 32 bits. Los formatos números de **Blackfin** se nombran N.M, siendo N la cantidad de bits dedicados al signo y M la cantidad de bits dedicados a la mantisa El **Blackfin** puede manejar los siguientes formatos númericos:

1. Entero de 32 bits.
2. Entero de 32 bits con signo.
3. Fracción de 32 bit sin signo (0.32).
4. Fracción de 32 bits con signo (1.31).

Por ello ha sido necesario realizar rutinas de adaptación especial para tres operaciones punto flotante a operaciones en punto fijo. Dichas rutinas trataran de aprovechar la capacidad de paralelismo limitado del **Blackfin**, en particular su doble MAC. Los datos de entrada siempre van a ser números punto flotante IEEE simple precisión.

Recordemos que estos números son almacenados de la siguiente forma:

$$N_{IEEE754_{SP}} = \text{seeeeeeeemmmmmmmmmmmmmmmmmmmmmmm}$$

siendo la codificación:

$$N = \begin{cases} (-1)^s 2^{exp-127}(1.mant)_{10} & si\ 0 < exp < 254 \\ (-1)^s 2^{-127}(0.mant)_{10} & si\ exp = 0 \\ 0 & si\ exp = 0\ y\ mant = 0 \\ +\infty & si\ s = 0,\ exp = 255\ y\ mant = 0 \\ -\infty & si\ s = 1,\ exp = 255\ y\ mant = 0 \\ NaN & si\ s = 0,\ exp = 255\ y\ mant \neq 0 \end{cases} \quad (7)$$

donde $exp = (eeeeeeee)_{10}$ y $mant = (mmmmmmmmmmmmmmmmmmmmmmm)_2$

Una importante referencia en implementación de operaciones punto flotante es [24], la cual se ha tomado como punto de partida para todas las operaciones, salvo la raiz cuadrada. Las operaciones implementadas son:

1. Cuadrado. Se mejora la rapidez del cuadrado creando una rutina con las siguientes condiciones:
   a) Exponente. Se realiza la operación al cuadrado
   b) Mantisa al cuadrado. Hay que realizar una multiplicación de punto fijo de 24 bits. Se cambia dicha multiplicación de 32 bits por 4 multiplicaciones de 16 bits, corrigiendo el punto decimal con lo cual se aumenta la eficiencia.
   c) Casos fuera de norma. Se integran al algoritmo general el tratamiento de los casos particulares.
2. Suma. El programa implementando consta de los siguientes pasos.
   a) Intercambio. Aprovechando la disposición del formato fp que permite comparar dos números en forma poco costosa, se ordenan los números de forma tal que el primero sea el de mayor valor absoluto. También se chequea si uno de los

números es 0 para retornar rápidamente de la operación en ese caso.

b) Exponentes. Se mira la diferencia de exponentes entre los números. Si la misma es mayor que la cantidad de bits disponibles para la mantisa (23) se retorna el mayor.

c) Desplazamiento. Para poder sumar dos mantisas deben estar refiridas al mismo exponente. Para ello se desplaza la mantisa del menor la cantidad de bits necesarios.

d) Suma algebraica. Según los signos de los números se realiza suma o resta de las mantisas.

item Casos extremos. Se realiza la detección y tratamiento de casos de overflow y underflow.

e) Normalización. Se lleva el exponente y la mantisa a la forma IEEE 754 precisión simple.

3. Resta. Se implementa como un caso particular de la suma, cambiando el signo del número aa restar y realizando la suma algebraica de los números.

4. Raiz Cuadrada. Se tratara de acelerar el cómputo de esta operación recurriendo únicamente a operaciones realizables en aritmética de punto fijo. En el caso general hay que encontrar: $\sqrt{N} = \sqrt{2^{exp-127}(1.mant)_{10}} = \sqrt{2^{exp+1-127}(0.mant)_{10}}$

   Sea $A = 2^{exp+1-127}$ y $B = (0{,}1mant)_{10}$ La raiz se calcula en dos partes:

   a) Raiz del exponente: $\sqrt{A} = 2^{\frac{exp+1-127}{2}}$

   - Si A es par $\Longrightarrow$ $\sqrt{A}$ se calcula por desplazamiento del exponente
   - Si A es impar $\Longrightarrow$ se calcula $\sqrt{A*2}$ por desplazamiento. Posteriormente al resultado de $\sqrt{B}$ se lo multiplicará por $\frac{1}{\sqrt{2}}$ para compensar el 2 agregado en A.

b) Raiz de la mantisa: $\sqrt{B} = \sqrt{(0{,}1mant)_{10}}$

Con las operaciones anteriores nos hemos asegurado que $0{,}5 \leq B \leq 1$

Esto implica que $0{,}25 \leq \sqrt{B} \leq 1$

Por Taylor: $f(x) \cong f(x_0) + (f'(x_0)(x - x_0) \Rightarrow \sqrt{x} \cong \frac{1}{2\sqrt{x_0}}x + \frac{1}{2}\sqrt{x_0}$

Para calcular $\sqrt{B}$ se divide el intervalo $[0{,}5, 1]$ en 16 sectores y se linealiza la ecuación. Se construye una tabla con los valores de $\frac{1}{2\sqrt{x_0}}$ y $\frac{1}{2}\sqrt{x_0}$ para cada uno de los intervalos. La decodificación del intervalo a que pertenece se realiza mediante los primeros 4 bits de B.

Todo queda reducido a operaciones de punto fijo sin overflow.

Posteriormente se normaliza el resultado

Los programas en Assembler son:

- *cuadradofp.S*
- *sumafp.S*
- *raizcuadradafp.S*
- *vector_dist_bf.S*

Además se ha modificado varios programas del SOM_PAK para que se utilicen los programas optimizados en Assembler

F.1 Evaluación de resultados

Para comparar la implementación de *SOM* en **Blackfin** con otra arquitectura, como por ejemplo una *PC* de escritorio se tomará un problema tipo de comparación.

Como problema de comparación se tomará el incluído con el paquete *SOM_PAK*, que consiste en los datos de estado de una máquina industrial recogidos a lo largo de 24h, los cuales tienen dimensionalidad 5. Se totaliza 3840 vectores. La matriz tiene una dimensión de 10 x 10.

Plataforma	Velocidad del Procesador	Memoria	Cantidad de patrones de entrada	Tiempo de entrenamiento
Blackfin bf537 sin Assembler	500 MHz	64 MB	3048	18 seg
Blackfin bf537 con Assembler	500 MHz	64 MB	3048	15 seg
Pentium IV dual core	3.2 GHz	1 GB	3048	5 seg

TABLE V
COMPARACIÓN DE TIEMPOS DE ENTRENAMIENTO

El tiempo de entrenamiento es medido con los datos proporcionados por el propio *vsom*. Se realizarón pruebas con tres tipos de sistemas: **Blackfin** con *SOM_PAK* sin optimización *Assembler*, **Blackfin** con *SOM_PAK* con optimización *Assembler* y *PC* de última generación.

Comparando la performance entre los dos sistemas tenemos que:

Observamos que se obtiene una mejora con las rutinas de Assembler en la placa **Blackfin**. Sin embargo la performance obtenida en la placa es inferior a la que se obtiene en el PC, pero es necesario tener en cuenta la diferencia de recursos entre ambas plataformas, compensado por el hecho que el procesador **Blackfin** cuesta en torno a USD 5, mucho menos que un *Pentium IV dual core*.

VII. Diseño de la aplicación anti *SPAM*

A. Introducción

En esta sección presentaremos el diseño de la aplicación *antispam* utilizada para demostrar las funcionalidades del Sistema SOM corriendo en el DSP. Basándonos en los conceptos analizados en el capítulo III, intentaremos que la aplicación pueda presentar una solución al problema del *spam*.

B. Arquitectura de la solución propuesta

El procesamiento de la información contenida en los mensajes de correo entrante, requiere de una serie de funcionalidades que debimos desarrollar. En principio fue necesario identificar la forma en que el programa cliente de correo electrónico **Evolution** [25] era capaz de solicitar el procesamiento de un *mail* entrante por un agente externo. Una vez resuelto, debimos implementar una serie de herramientas que permiten extraer la información que nos interesa del mensaje. Dicha información deberá ser procesada y luego enviada a la red neuronal que se encargará de crear el Mapa Auto-organizado para finalmente lograr clasificar los *mails*.

La obtención de los *mails* que llegan al programa de correo **Evolution**, se logró mediante el uso de filtros de correo disponibles en la interfaz del usuario (Fig. 9). Dichos filtros mantienen en espera el mensaje antes de depositarlo en alguna carpeta, al tiempo que lo disponibilizan en la salida estándar *STDOUT*, para su lectura o procesamiento por uno o varios programas de detección de *spam*. **Evolution** queda aguardando un *Score de spam*, y permite al usuario decidir que acción tomar. Ésta técnica que aplicaremos es la misma utilizada por otros productos comerciales como es el caso de *SpamAssassin* [26]. Es importante señalar, que el mensaje no se pierde, simplemente es demorado en su presentación al usuario. Podría perderse únicamente si configuramos intencionalmente en la regla la eliminación del mensaje.

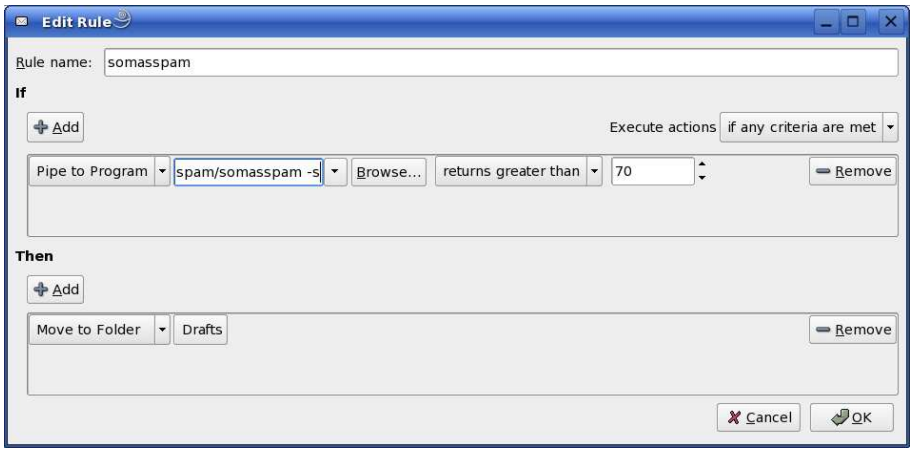

Figura 9. Configuración de filtros en *Evolution*

1. *¿Qué analizar?*

Luego de obtenido el mensaje en su totalidad nos planteamos descubrir que datos podrían ser relevantes para realizar la codificación del mensaje a valores numéricos y posibilitar la detección de *spam* usando la red neuronal [27].

Un mail SMTP consiste usualmente de dos partes: **el cuerpo del mensaje** normalmente compuesto por texto en lenguaje natural, posiblemente código HTML y gráficos, y **el encabezado** del mensaje. Este último es un conjunto estructurado de campos. Algunos de ellos son estándares como el *From* (De), *To* (Para) y *Subject* (Asunto), mientras que otros dependen del software utilizado para la transmisión del mensaje.

El campo *Subject* contiene el título del mensaje y muchas veces es tratado como parte del cuerpo del mensaje. El cuerpo del mensaje es considerado como el contenido del mensaje.

Para cada método de análisis de mensaje es necesario seleccionar cuál característica o parte del mensaje puede ser relevante.

La manera mas simple consiste en considerar el cuerpo del mensaje como una “Bolsa de Palabras” que permiten caracterizar una parte del mensaje o su totalidad. Sin embargo, existen

propuestas mas complejas que consideran el comportamiento de los *spammers*, identificando por ejemplo campos *From* vacíos, o fechas incorrectas [28]. También existen trabajos que consideran como datos para la clasificación, el tamaño del mensaje, remitente y recipiente del mismo (*sender y mailbox*), dominio y cantidad de archivos adjuntos.

2. *¿Cómo analizarlo?*

Los primeros filtros consistían en buscar la presencia de ciertos *tokens* o palabras claves en el cuerpo del mensaje, o verificando la existencia del remitente en alguna lista de posibles *spammers* (*blacklist/whitelist*).

La mayoría de los filtros de *spam* existentes son basados en métodos de aprendizaje. A partir de 1998 es propuesto el modelo clasificador de Naïve Bayes como herramienta de filtrado por palabras claves. Actualmente existen trabajos relativos a la utilización de SVM (*Support Vector Machines*) [29] para filtrado.

Mientras que muchos métodos son aplicables a la clasificación de texto en general, los métodos basados en la estructura del mensaje en sí, usan técnicas de análisis de ciertas características. Como ejemplo exiten filtros basados en el análisis de direcciones IP en camino inverso, asociando un grado de reputación del mismo. Otras soluciones analizan el campo *From, To, CC y Bcc*.

La figura 10 resume las técnicas de análisis de correo electrónico.

Para todos los algoritmos de clasificación existe el mismo problema, encontrar una solución para dos tipos de errores: clasificar mail válido (*ham*) como *spam*, es decir el índice de falsos positivos debe ser bajo, o clasificar *spam* como mail válido (*ham*), o sea que es necesario tener un bajo índice de falsos negativos. Algunas soluciones propuestas consisten en sistemas de colaboración entre usuarios, o reportes desde los usuarios a servidores de mails

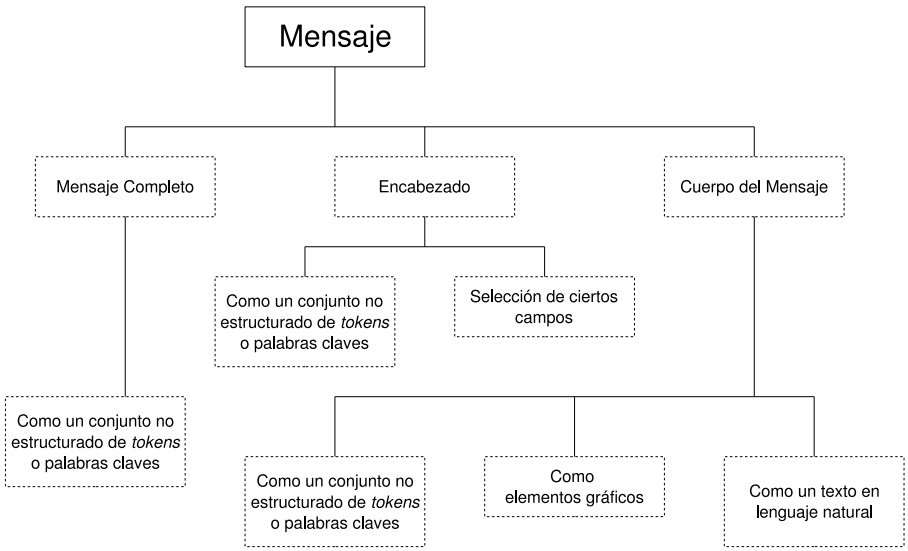

Figura 10. Técnicas de Análisis de Correo Electrónico

(Ej, Google Gmail).

3. *Reacciones opuestas*

Los métodos de los *spammers* son renovados a la vez que se crean nuevos sistemas de filtrado. Por ejemplo, modificando características anteriormente usadas, colocando caractéres de espacio extras en las palabras, introduciendo nuevas combinaciones de los mismos (Ej: Vi@6r@ en lugar de VIAGRA) o realizando ataques de *spam* para aumentar cierto tipo de estadísticas en mensajes de una categoría específica (*strong*) o ataques aleatorios (*weak*).
La reacción de los *spammers* requiere contramedidas de los desarrolladores de filtros de *spam* y de los administradores de redes. Precisamos una solución adaptable a los cambios en la modalidad de *spam*.

B.1 Solución adoptada

Intentamos abarcar algunos de los conceptos estudiados para el análisis de los mensajes de correo los cuales se pueden resumir en los siguientes:

Análisis del Encabezado del mail.

- From (user, domain)
- To
- Bcc
- IP

Análisis del Contenido del mail.

- Asunto: búsqueda de palabras claves
- Cuerpo del Mensaje: Estudio del contenido temático

Para el encabezado consideramos más importante los datos del remitente que de los destinatarios, y para el contenido nos basamos en algunas investigaciones previas [27] que arrojan resultados más favorables si analizamos el asunto, que si analizamos el cuerpo del mensaje. Por lo tanto pasaremos a analizar:

- Descripción de remitente (por lo general nombre y apellido)
- campo *user* de la dirección del remitente
- campo *domain* de la dirección del remitente
- dirección IP del *host* o servidor de correo del remitente
- Asunto (*Subject*) del mail

Con estos cinco (5) datos llegaremos a un conjunto de valores de codificación de los mensajes que servirán tanto para la función de entrenamiento de la red, así como para la clasificación de los mensajes entrantes.

Cabe destacar que se intentó simplificar la aplicación no incluyendo más datos, como por ejemplo el texto del cuerpo del mensaje, debido a la complejidad del análisis del mismo y basándonos en anteriores investigaciones [10] que arrojan resultados similares para filtros que analizan *Asunto* contra otros que analizan el *cuerpo del mensaje*. La solución óptima podría

ser analizar los dos campos, desarrollo que podríamos dejar abierto para futuras mejoras del proyecto.

C. Implementación de la solución

Nos proponemos crear un sistema capaz de funcionar en un ambiente de redes de computadores como servidor *Antispam*. La figura 11 muestra la configuración general del sistema. Como vemos será necesario lograr que nuestro sistema se comunique con los distintos clientes de la red, para lo cuál utilizaremos Sockets TCP/IP. El acceso a los datos en recursos compartidos se hará mediante el protocolo NFS. Esto nos permitirá tener independencia en

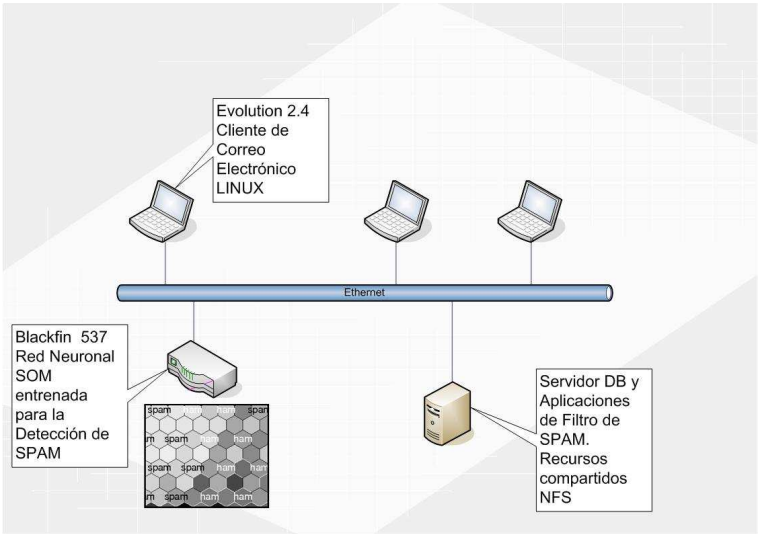

Figura 11. Diagrama General del Sistema

cuanto al tamaño del archivo de datos de mails para el entrenamiento del sistema. Y a su vez nos dará la flexibilidad de implementarlo para uno o varios clientes.

En su forma más simple, la solución estará compuesta esencialmente por dos sistemas funcionando en distintos *hosts* de la red:

- *El Cliente de Correo*

- *El Servidor SOM*

En el *Cliente de Correo* va a estar la funcionalidad necesaria para extraer los datos del mensaje, su codificación y almacenamiento, o envío al *Servidor SOM*.

Por otro lado, en el *Servidor SOM* tendremos corriendo una aplicación que recibe la codificación ya sea de la totalidad de los mensajes recibidos hasta el momento de la etapa de entrenamiento, o bién la codificación de un un mensaje individual para su clasificación. Pasaremos a describir los procesos que están disponibles en el sistema

C.1 Procesos en el Cliente de Correo

La aplicación del lado del cliente cuenta con dos procesos:

- ***trainsom*** contiene las funciones para preparar el entrenamiento de la red neuronal
- ***somasspam*** realiza la comunicación con la red neuronal y clasificación de mensajes entrantes

C.2 Procesos en el *Servidor SOM*

- ***som*** entrena la red en el Servidor SOM, queda a la espera de llamadas para clasificación de mensajes

El esquema de los procesos se puede observar en las figuras 12 y 13

D. Descripción de los bloques de software de la solución

Como vimos anteriormente, la solución planteada se puede dividir en 3 grandes bloques de procesamiento:

1. El programa de codificación de los mensajes del cliente: ***trainsom***
2. El programa de envío de la codificación al *Servidor SOM*: ***somasspam***

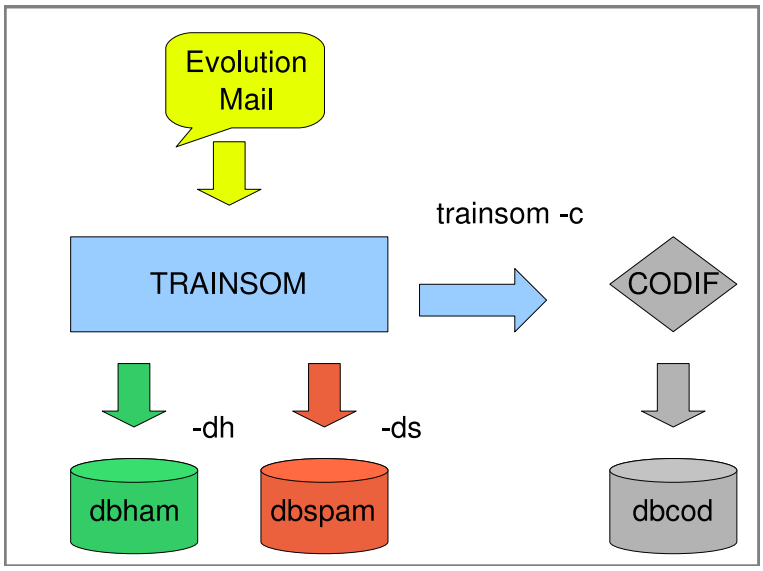

Figura 12. Proceso *trainsom*

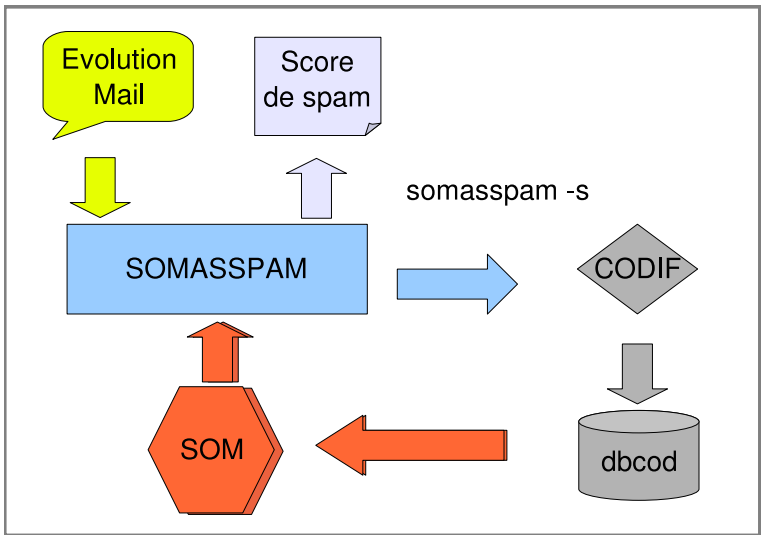

Figura 13. Proceso *somasspam*

3. El sistema ejecutandose en el *host* remoto o *Servidor SOM* para procesamiento de la codificación: ***som***

D.1 Programa de codificación de los mensajes del cliente: *trainsom*

Este programa es capaz de analizar el contenido de un mensaje de correo electrónico extrayendo los datos necesarios para su posterior codificación. El mensaje puede encontrarse

en un archivo de texto plano o ser tomado de la salida estándar *stdout*. Este último caso es el utilizado por el **Evolution** mediante un mecanismo de *pipe* o entubado hacia el programa ***trainsom***. Una vez recibidos los campos del mensaje y de alimentar la base de *ham* y *spam* respectivamente, ***trainsom*** ejecuta ciertas rutinas programadas en *codif.c* que transforman dichos campos en un conjunto de 5 valores decimales con precisión 1×10^{-12}. Estos valores son almacenados en una base llamada *dbcod* y *dbcod_lab* para el futuro entrenamiento y etiquetado de la red neuronal.

D.2 Programa de envío de la codificación al *Servidor SOM*: *somasspam*

Análogamente a su par *trainsom*, ***somasspam***, realiza lectura de campos y codificación de los mismos, pero ahora ya no para alimentar una base, sino que para consultar al **Servidor SOM** sobre cuál es la neurona mas próxima (neurona ganadora) en el Mapa SOM para estos valores de codificación que conforman el vector de entrada. Luego de obtenido ese dato realiza un cálculo del *Score* de *spam* con la ayuda del archivo de visualización del mapa (Fig. 14) *dbcod_lab.ps* que está en formato *PostScript*.
Otra función de este programa es ser capaz de solicitar al *Servidor SOM* el reentrenamiento de la red sin necesidad de reiniciar la aplicación en el *host* remoto.

D.3 Sistema ejecutandose en el *Servidor SOM* remoto: *som*

Éste programa optimizado para la plataforma de desarrollo DSP, una vez iniciado verifica si es necesario entrenar la red neuronal o simplemente toma el archivo del mapa entrenado en alguna ejecución anterior. Se comunica a travez de la Interfaz Ethernet de la placa de desarrollo, y mediante *Sockets* abre un puerto TCP (el puerto 3490 debe estar abierto en el *Firewall*) y queda a la espera de las conexiones provenientes de cualquier *host* cliente. El cliente proporciona un vector de entrada de dimensión igual a los vectores de entrenamiento

del mapa SOM, y el Servidor devuelve otro vector que contiene las coordenadas de la neurona ganadora para ese vector en dicho mapa, y el error de cuantización obtenido en el cálculo. Como se dijo anteriormente, el *Servidor SOM* es capaz de recibir conexiones de distintos clientes que pueden o no haber aportado datos para el entrenamiento de la red. Lo que en un ambiente de Red de Área Local significaría tener un servidor destinado a utilizar todos los datos de entrenamiento de los clientes, para ofrecer un servicio de clasificación de mensajes.

D.4 Parámetros inherentes a la estructura de la red neuronal

La aplicación ***som*** que corre en la placa de desarrollo es totalmente configurable mediante un archivo de configuración editable. En la misma podemos elegir el tamaño de la red a entrenar, el tipo de función de interacción lateral, o función de proximidad y la orden de entrenamiento al inicio de la aplicación. Esto nos permite a medida que crece el volumen de datos, reentrenar la red con parámetros mas efectivos para la aplicación.

```
#DIMENSIONES DEL MAPA
[X_DIM]  10
[Y_DIM]  10
#TIPO DE DISTRIBUCION DE LA FUNCION DE PROXIMIDAD bubble / gaussian
[DISTRIB] bubble
#RE-ENTRENAMIENTO DE LA RED AL INICIO DE LA APLICACION yes / no
[TRAINNING]  yes
```

Si bien el paquete SOM_PAK permite seleccionar entre dos estucturas de conexionado de las neuronas, es decir rectangular o euclideana, y hexagonal, debido a las pruebas hechas en las etapas de desarrollo y a otros experimentos similares, se optó por utilizar un topología

hexagonal. Esta configuración en prinicipio no es parametrizable, ya que para el desarrollo de la aplicación de clasificación de correo se requiere especificar de antemano la topología utilizada.

D.5 Parámetros inherentes a la funcionalidad anti *spam*

Existen varios parámetros configurables en el cliente, el más importante y sensible a los resultados de la aplicación es el límite inferior que fijamos para determinar si un mensaje es *ham* o *spam*. El rango configurable va de 0 a 100 respectivamente, y la aplicación ***somasspam*** devolverá al **Evolution** un valor en ese rango para cada *mail* procesado. Esta facilidad le posibilita al usuario hacer mas o menos restrictivo el filtro de *spam* aumentando o disminuyendo el umbral.

La aplicación del lado del cliente, cuenta con un archivo de configuración *somasspam.config*, para definir el directorio donde se encuentran las bases de entrenamiento y codificación de *ham* y *spam*, y el nombre del *Servidor SOM* con el que se comunicará.

E. Notas sobre el diseño

Trataremos de hacer una síntesis de los programas desarrollados en nuestro sistema haciendo incapié en aquellos más sensibles a los resultados de la aplicación. Dichos programas se focalizan en realizar una correcta codificación que facilite el entrenamiento y clasificación por intermedio del Mapa SOM.

A continuación describiremos los archivos de programa y archivos generados por el sistema, directorios y recursos compartidos donde deben estar.

Finalmente haremos un breve comentario sobre la comunicación entre las distintas partes del sistema, las posibles mejoras y la integración con otras herramientas de detección de *spam*.

E.1 Codificación de los mensajes

Como fue mencionado anteriormente, debemos codificar los siguientes datos:

- Descripción de remitente (por lo general nombre y apellido)
- campo *user* de la dirección del remitente
- campo *domain* de la dirección del remitente
- dirección IP del *host* o servidor de correo del remitente
- Asunto (*Subject*) del mail

1. Codificación de la descripción de remitente y del campo *user* del *mail*

De la observación de gran cantidad de mensajes de *spam*, surge la idea de intentar buscar alguna relación entre estos dos datos, encontrandonos que en la gran mayoría de los mensajes de *spam*, no existe correlación alguna. Esto es posiblemente, debido a la forma en que los *Spammers* generan el *spam*. Las direcciones de correo por lo general son conjuntos aleatorios de caractéres, mientras que los nombres de los remitentes intentan ser más amigables o son tomados de los *Address Books* de los usuarios. Se codifica mediante un algoritmo que analiza carácter por carácter el campo *user*, y lo busca en la descripcion del remitente. Luego le asigna un peso que depende del largo del *user* si encuentra ese carácter, y cero si no lo encuentra. Así llegamos a un valor de correlación para estos dos datos.[7]

2. Codificación del *dominio* del remitente

Para este dato, intentamos observar la procedencia de los *spams*, gran cantidad de ellos parten de servidores de correo públicos, *hotmail.com*, *yahoo.com*, etc. Otra gran mayoría de dominios extraños como lo serían *luckybux13.com*, *click-4-cash.com*, etc. Y gran cantidad de

[7] Function *correl_str(user,descFrom)* de caracteres.c

ellos vienen con código de países exóticos: *.ru*, *.do*, *.au*, *.hr*, etc.[8]

Debido a este hecho, y para diferenciarlos claramente de dominios conocidos recibidos por el usuario, se codificó cada carácter del dominio con una cifra decimal de una tabla predefinida, que se suma al siguiente carácter llegando a un valor decimal que representa en su parte más significativa al código de país, y luego al dominio.[9]

3. Codificación del *IP* del servidor remitente

De manera de acentuar el concepto anterior, y dado que podrían existir conjuntos de IPs más utilizadas por los *Spammers*, se procedió a convertir cada uno de los cuatro (4) octetos de la dirección IP, en un valor decimal que la representa: simplemente se dividió entre 100 o 1,000 cada octeto de acuerdo a si era menor o mayor que 100 respectivamente, y luego se concatenó al siguiente, quedando en la posición mas significativa el primer octeto.[10]

4. Codificación del *Subject*

Para este último dato y dada su sensibilidad, nos concentramos en buscar técnicas de clasificación de textos. Encontramos que la mayoría de ellas están focalizadas en textos de gran tamaño y variedad de temáticas o categorías. Existen en la bibliografía estudios sobre clasificación de textos que requieren un gran procesamiento de palabres. En el caso de la detección de *spam*, simplemente debíamos reconocer palabras frecuentes en este tipo de mensajes para luego compararlas con textos normales.

Debimos implementar diccionarios de palabras típicas en mensajes de *spam* y en mensajes de *ham*, asociando a cada término un valor que representa la cantidad de veces que aparece un término en todo el conjunto de mails de aprendizaje, es decir la frecuencia.[11]

[8] .uy será exótico para la China ;-)
[9] Sección *codif domain* de codif.c
[10] Function *codif_ip* de codif.c
[11] Function *creaDic* de caracteres.c

Luego para codificarlo, encontramos muy utilizada la fórmula de cálculo de la Frecuencia Inversa del término para ponderar cada palabra del *Subject*:

$$IDF = log(\frac{D}{Dw_i})$$

Siendo D el tamaño del Diccionario de *ham* o *spam*, y Dw_i la cantidad de veces que aparece el término en el asunto en cuestión.

En algunas pruebas, fue ventajoso utilizar también la suma de la frecuencia relativa de cada término en cada diccionario.

$$\text{Valor de codificación} = \Sigma Dw_i * (\frac{Qw_i}{D})$$

Siendo Qw_i la cantidad de veces que aparece el término en todo el conjunto de mails de entrenamiento.[12] Pero finalmente optamos por el primer criterio de codificación.

E.2 Almacenamiento de datos - Formato de los Archivos

Los archivos generados por el sistema en orden de aparición en el directorio del Cliente son:

1. Archivos base

Primer grupo de archivos utilizado para guardar los cinco datos de cada mail seleccionado para entrenamiento, y el diccionario de *ham* , y *spam*.

- ***dbham.dat***
- ***dbspam.dat***
- ***dictham.dat***

[12] Function *calc_dic* de codif.c

- ***dictspam.dat***

Un fragmento de cada archivo podria ser:

```
dbham.dat:
albertomariani noreply@yahoogroupscom 66.218.66.71  IC Re Consejo acerca el campo
    laboral en Redes de Computadoras

dictham.dat:
__dic_ham_version_1.0 0
...
actividad 1
acerca 1
activ 1
...
contrata 1
consejo 4
conocidos 1
...

dbspam.dat:
VINA Langman solubledefectives@dolnet 111.67.142.60  VINA  Viagra for you


dictspam.dat:
__dic_spam_version_1.0 0
```

```
...
vatican 2
Viagra 21
Viktoria 4
VIENE 4
...
Zack 4
yourself 4
you 5
yarmouth 3
...
```

Como se puede observar tenemos por un lado los archivos con los datos necesarios para la clasificación, y por otro los archivos de diccionario para cada tipo de mensaje, donde aparece el término y la cantidad de veces que se encontró en todos los mensajes seleccionados para el entrenamiento.

2. Archivos Codificados

A través ***trainsom*** se generan los archivos codificados, en dos versiones, sin etiquetas y con etiquetas:

- ***dbcod.dat***
- ***dbcod_lab.dat***

```
dbcod.dat:
5
0.500000 0.46158158 0.20023315 0.00162866 0.00000000
```

```
0.625000 0.43076574 0.69701022 0.64983714 0.02906977

0.750000 0.46267442 0.17216119 0.11563518 0.02906977

0.777778 0.47358557 0.20121719 0.00000000 0.00000000

0.866667 0.54313570 0.12700100 0.51302934 0.04360465

...
```

```
dbcod_lab.dat:

5

0.428571 0.43077008 0.66218668 0.67426711 0.02906977 ham

0.333333 0.46158158 0.19216801 0.30456027 0.01744186 ham

0.750000 0.51586744 0.20122139 0.02605863 0.00436047 ham

0.250000 0.66372758 0.19424242 0.00000000 0.02180232 spam

0.111111 0.57237841 0.65185118 0.00000000 0.00000000 spam

0.125000 0.51441576 0.88764226 0.00325733 0.10755814 spam

0.500000 0.51608966 0.87103182 0.00000000 0.00145349 spam

...
```

Una vez iniciada la aplicacion ***som***, la misma toma los archivos recientemente codificados y genera el mapa SOM en archivo de texto, y en formato PostScript.

- ***dbcod.cod***
- ***dbcod_lab.ps***

```
dbcod.cod:

5 hexa 10 10 bubble

0.191588 0.500993 0.869048 0.0197603 0.046246 spam

0.240609 0.507738 0.857717 0.0179667 0.0456153 spam
```

```
0.655609 0.537935 0.179192 0.106604 0.0379771 ham
0.775268 0.524252 0.175276 0.119409 0.0297128 ham
0.0854043 0.528869 0.792596 0.0160601 0.0618102 spam
...
```

El cual contiene en su encabezado la dimensión de los vectores de entrada, las dimensiones del mapa y el tipo de función de interacción lateral.

A continuación prosiguen los registros de los pesos de cada neurona del mapa.

El archivo de visualización del mapa (Fig. 14) nos muestra las neuronas del mapa etiquetadas como *ham* en color blanco y como *spam* en color negro, o sin etiquetar aquellas que en el entrenamiento no resultaron ser neuronas ganadoras por votación para algunas entradas.

Además en el interior de la neurona en escala de grises se representa la proximidad a sus neuronas vecinas:

- Mas claro = Neuronas más próximas
- Mas oscuro = Neuronas más apartadas

Como dato extra, el algoritmo *umat* dibuja hexágonos separando una neurona de las otras representando así en escala de grises la distancia entre ellas. Dicho programa genera un archivo *PostScript* que contiene está información codificada en forma matricial. Esta representación grafica de la red SOM se demonina *U-matrix*.

Cabe aclarar que en esta representación el valor de gris asociado al hexágono correspondiente a una neurona del mapa (no a un hexágono de separación) se obtiene calculando la mediana de la distancias a las neuronas vecinas.

La forma en que SOM_PAK codifica la distancia en la escala de grises es la siguiente:

1. Calcula todos las distancias entre dos neuronas vecinas, siendo el concepto de neuronas vecinas distinto según el tipo de vencidad. Para la propia neurona se calcula la

mediana de las distancias a las neuronas vecinas.

2. Obtiene el valor máximo (MAX) y mínimo (MIN) de las distancias.
3. Mediante una función afín (líneal con *offset*) de pendiente negativa se escala el conjunto de distancias al intervalo $[0, 1]$ con el criterio que 0 corresponde a MAX y 1 corresponde a MIN.
4. Al construir el archivo *PostScript* los valores de grises se calculan multiplicando las distancias por 100 y tomando la parte entera, con lo cual las distancias quedan escaladas al intervalo $[0, 100]$.

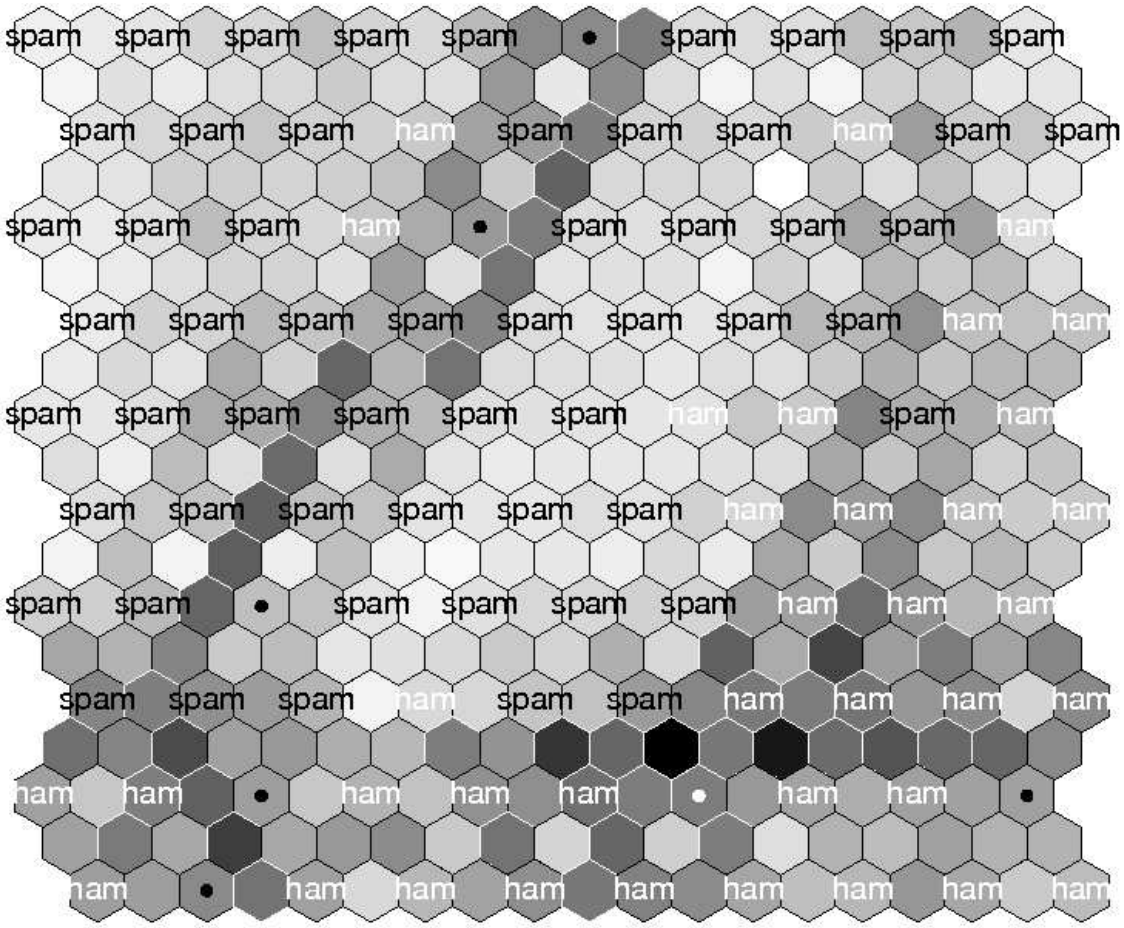

Figura 14. Mapa SOM entrenado para la detección de *spam*

E.3 Cálculo del *Score de Spam*

Llegado a este punto de conocimiento del funcionamiento del sistema, podemos discutir la fórmula para el cálculo del *Score de Spam*.

La Red Neuronal SOM está entrenada en base a un conjunto de *mails* conocidos y previamente clasificados como *spam* o *ham*. Este conocimiento previo nos permitirá calcular para cada *mail* entrante que zona de la red le corresponde. El algoritmo ***visual*** de **SOM_PAK**, realiza dicho cálculo, para cada vector de entrada de dimensión n que se proporcione devuelve las coordenadas de dicho vector en el mapa bidimensional, junto con un valor de error de cuantización obtenido en el cálculo. Una primera aproximación para asignar un *Score de Spam* a dicho vector, sería verificando la etiqueta de la neurona ganadora. Pero, ¿qué sucede si se trata de una neurona no etiquetada?

Para evitar zonas de indefinición, creamos una función [13] que considera la etiqueta de la neurona ganadora en caso de tenerla y además lo que sucede en su entorno más cercano en el mapa (Fig. 15).

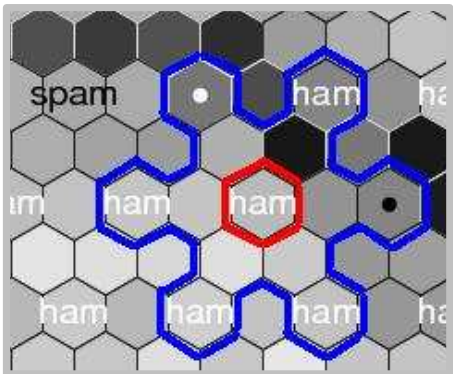

Figura 15. Neurona ganadora y su entorno en el Mapa SOM

El cálculo se realiza de la siguiente manera, se parte de dos variables enteras que contabilizan el peso de *ham* y de *spam* de dicho vector de entrada:

[13] Function *calcScore()* de codif.c

- $w_ham = 0$
- $w_spam = 0$

y de un resultado que en forma preliminar asigna el valor 0 si la neurona ganadora no está etiquetada, 100 si la etiqueta de la neurona ganadora es *spam* o -100 si es *ham*.

- $result = 0$, Si la neurona ganadora no está etiquetada
- $result = 100$, Si la neurona ganadora tiene etiqueta de *spam*
- $result = -100$, Si la neurona ganadora tiene etiqueta de *ham*

Inmediatamente es ponderado dicho valor por el error de cuantización del cálculo mediante la siguiente fórmula:

$$result = (int)(100 * (1 - q_error))$$

Considerando la forma hexagonal de la red en la representación *U-matrix*, para recorrer las neuronas vecinas el algoritmo deberá hacer 6 cálculos, uno para cada dirección de la neurona ganadora. Los pesos de *ham* y *spam* se verán incrementados de acuerdo a la cantidad de neuronas vecinas con *ham* o *spam* en sus etiquetas. El valor de incremento dependerá del grado en la escala de grises, que va de 0 para más oscuro y 100 para más claro, es decir de más apartado a más cercano.

Gris claro = Valor próximo a 100 = Neuronas más cercanas

Gris oscuro = Valor próximo a 0 = Neuronas más apartadas

Para ello convertimos la matriz *mxn* dada en el archivo *PostScript* a una matriz de igual dimensión pero de estructurados que contienen dos campos Etiqueta y Escala de Gris. Cada componente de dicha matriz será de la forma:

$$umatrix[i][j].label$$

$$umatrix[i][j].gray_lev$$

Si una neurona vecina está cerca, o sea con intensidad de gris claro en la propia neurona y en la neurona intermedia de separación, su etiqueta pesará más que otra que se encuentre más apartada. El cálculo que se realiza para cada neurona de la vecindad es:

Si la etiqueta de la neurona vecina es *ham*, $w_ham = w_ham - umatrix[i][j].gray_lev$

Si la etiqueta de la neurona vecina es *spam*, $w_spam = w_spam + umatrix[i][j].gray_lev$

Este cálculo se realiza de manera de recorrer las seis neuronas que rodean a la neurona ganadora, teniendo en cuenta que en los bordes del mapa sólo se puede recorrer las neuronas internas al mismo, para lo cuál se tiene un contador *cont* de neuronas vecinas, estén etiquetadas o no. Finalmente se promedia el valor del resultado preliminar de la neurona ganadora, junto con los pesos de *ham* y *spam* de las neuronas vecinas

$$result = (w_ham + w_spam + result)/(cont + 1)$$

En el mayor de los casos, si la neurona ganadora y las vecinas tienen la misma etiqueta, y además la separación es mínima, o sea con escala de gris próximo a 100, serán 13 términos los que se suman al mismo tipo de *mail*. Dichos terminos corresponden a: la neurona ganadora, las 6 neuronas vecinas etiquetadas y las 6 neuronas de separación intermedia. Se llega a un valor próximo a 1300 para *spam* o -1300 para *ham*. El promedio estará comprendido entonces en el rango de -100 a 100.

Finalmente es necesario hacer un cambio de escala [-100, 100] para [0, 100] tal cual lo utiliza el cliente de correo electrónico **Evolution**. Es decir que si la red está bién entrenada y el resultado de la clasificación es mayor que 50 se trata de *spam* si es menor que 50 se trata de *ham* en caso de haber hecho un ajuste medio del umbral. Como veremos luego, el usuario podrá ajustar el umbral de manera deseada para hacerlo más o menos restrictivo al pasaje

de *spam*.

E.4 Ejemplo de cálculo de *Score de Spam*

El siguiente ejemplo muestra el resultado de la aplicación del filtro *Somasspam* a un *mail* cualquiera que llegó al cliente. En este caso un *mail* de *secre@cei.fing.edu.uy* dirigido a una lista de destinatarios con el asunto: *[todos] licencia x estudio —- y censo*. Luego de aplicado el filtro *Somasspam* y de acuerdo al archivo de log del sistema *somasspam.log*, verificamos que la posición del neurona ganadora para ese mail y su *Score de Spam* fué:

```
>> mail >> secre at ceifingeduuy >> Position SOM: 9 6 0.0829007 >> Score Spam: 40
```

La coordenada x en el mapa $10x10$ va de 0 a 9, o sea que está en el borde respecto de las x, y la coordenada y se incrementa de arriba hacia abajo también de 0 a 9, está en la posición 6. La figura 16 muestra la ubicación.

La neurona ganadora está etiquetada como *ham* y se encuentra rodeada por una de *spam*, dos de *ham* y dos sin etiquetar que no aportan datos al cálculo. Comencemos desde abajo y en sentido horario a recorrer la vecindad de la neurona ganadora. Abajo las dos neuronas sin etiquetar no aportan datos, luego encontramos la primer neurona etiquetada como *ham* con una intensidad de gris de 63, y su neurona intermedia en 79. Estos datos tomados del archivo *dbcod_lab.ps* como se dijo anteriormente.

El valor temporario de peso de *ham* es $w_ham = -63 - 79 = -142$. Luego encontramos otra neurona de *ham* con una intensidad de gris de 88, y su neurona intermedia en 82. El valor temporario de peso de *ham* se incrementa ahora en $w_ham = -142 - 88 - 82 = -312$.

Finalmente, arriba se tiene una neurona etiquetada como *spam* con una intensidad de gris de 91, y su neurona intermedia en 91, con lo cual el valor temporario de peso de *spam* es $w_spam = 91 + 91 = 182$.

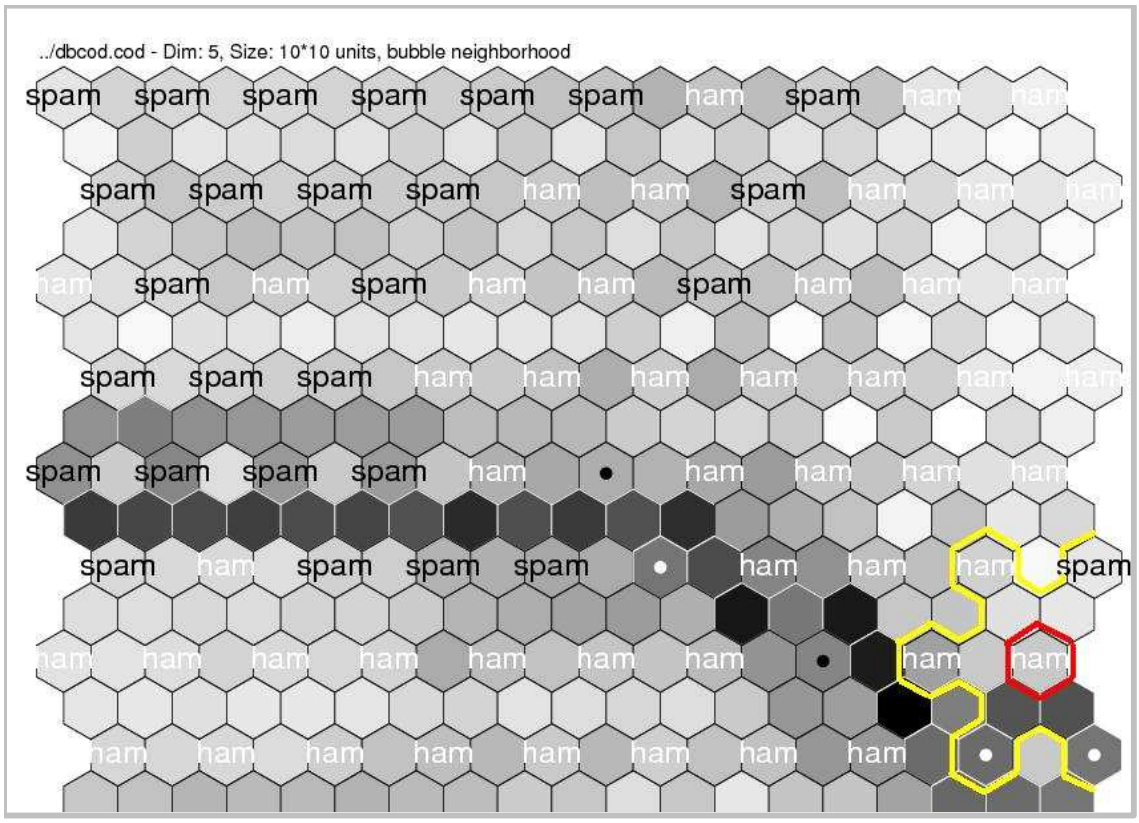

Figura 16. Neurona ganadora y su posición en el Mapa SOM

El resultado preliminar fue

$$result = (int)(-100 * (1 - 0{,}0829007)) = -91$$

El conteo de neuronas vecinas llegó en este caso a 5, que se duplica debido a las celdas intermedias, por lo tanto $cont = 10$.

El total de Score en la escala de -100 a 100 es:

$$result = (int)(-91 - 312 + 182)/(cont + 1) = -20$$

Con el cambio de escala:

$$score = (int)(score + 100)/2 = 40$$

Obtuvimos de esta forma un valor de *Score de Spam* inferior a 50, por lo tanto el *Evolution* dejará pasar este *mail* sin enviarlo a la carpeta de *Spam*.

3. Archivos auxiliares en el servidor SOM

Se trata de archivos temporales para el procesamiento de la función *visual* del paquete SOM_PAK.

- ***/tmp/tmp.dat˜***
- ***/tmp/tmp2.dat˜***

4. Archivo de *log*

Puede ser necesario observar el comportamiento del sistema, para lo cual se tiene un archivo que registra todo lo procesado.

- ***somasspam.log***

Cada línea del archivo indica el orígen del mail, la posición de neurona ganadora que obtuvo junto con su error de cuantización en el cálculo de la misma, y el *Score de Spam* asociado a ese *mail* (de 0 a 100).

```
>> mail >> secre at ceifingeduuy  >> Position SOM: 2 0 0.192721  >> Score Spam: 26
```

E.5 Archivos de Programas

Los archivos de programas desarrollados para la aplicación fueron:

- ***trainsom.c***
- ***spamassom.c***
- ***som.c***
- ***caracteres.c***
- ***dict.c***

- ***codif.c***
- ***cliente.c***
- ***server.c***

1. *trainsom.c*

Las opciones de este programa son las siguientes:

-dh stdin Lee mails de entrada std para base de HAM

-ds stdin Lee mails de entrada std para base de SPAM

Este programa tiene la función de preparar la información contenida en el programa de correo **Evolution** separando los datos necesarios para luego codificarlos y procesarlos con el algoritmo SOM. Para crear esta base, el programa tiene una opción *-dh* que simplemente guarda los campos a considerar de cada mail en un archivo de base de datos *dbham.dat* si es correo deseado. Para crear la base de *spam* la opción es *-ds* generando el archivo *dbspam.dat*. A medida que completa la carga de archivos, el programa realiza la creación de los diccionarios de *ham* y *spam*. Con estos diccionarios y los datos recabados anteriormente realiza la codificación de toda la base de *ham* y *spam*, para finalmente poder ejecutar el algoritmo **SOM** para el entrenamiento de la red.

2. *spamassom.c*

Este programa debe clasificar como *spam* o *ham* a los mensajes que vienen llegando . La función básica consiste en tomar de la salida estándar *stdout* el mail recibido en **Evolution** y capturar la descripción del campo *From*, el mail del remitente (usuario y dominio) y el *Subject*. Luego codificar dichos campos y enviarlos al algorimo SOM. Esta función se ejecuta

con los comandos:

- *spamassom -s* toma mails de la entrada standard
- *spamassom -m* filename verifica un mail guardado en un archivo
- *spamassom -t* envía la orden de reentrenamiento de la red

3. *som.c*

Este programa incluye todas las librerías necesarias para el procesamiento de las funciones definidas en SOM_PAK. Está codificado en C en su versión para PC, y en C / Assembler en su versión para el DSP. También incluye las librerías necesarias para ejecutar las funciones de *server.c*, es decir los Sockets de TCP, descriptos más adelante.

4. *caracteres.c*

Contiene la funciones necesarias para el manejo de Strings, así como las de generación y uso de los diccionarios de *ham* y *spam*. También posée las rutinas de interpretación de comandos de las aplicaciones.

5. *codif.c*

Contiene la funciones necesarias para la codificación de los datos extraídos de cada mail. Así como el código necesario para el cálculo de *Score de Spam* de un *mail*, dados el mapa SOM entrenado y la neurona ganadora para ese *mail* en particular.

6. *dict.c*

Contiene la definición de clases necesarias para el manejo de Diccionarios en estructura de árboles. Está codificado en C++ lo cual facilita la implementación de este tipo de estructura de datos.

7. *cliente.c*

Contiene las librerías para la implementación de Sockets TCP, en particular para abrir un Socket que permita enviar un *string* al servidor SOM, conteniendo los valores de codificación de cada mail, y luego recibir las coordenadas de la neurona ganadora y el error de cuantización en el cálculo. Todo esto lo realiza con una única función llamada *SockClient*.

8. *server.c*

Finalmente, este archivo tiene programado en un bucle, para el Servidor SOM, las rutinas de apertura del puerto TCP 3490, y espera de una o más conexiones a través de Sockets TCP. Las mismas implementadas con *fork* para cada nueva conexión (*vfork* para el DSP) . Los *strings* recibidos son procesados con las funciones de SOM_PAK *linkeadas* en este programa.

E.6 Comunicación de la aplicación anti *spam* con la red *som*

La comunicación del Blackfin con el PC se realizará de dos formas:

1. Funcionamiento en régimen. Cada patrón a clasificar se transmitira por medio de un Socket TCP en el puerto 3490. Para realizarlo se implementará el Socket servidor en la placa Blackfin y el Socket cliente en la PC. El Socket servidor podrá recibir pedidos de mas de un cliente, con lo cual varias máquinas de la red LAN podrán utilizar este mecanismo anti*spam*.
2. Archivos de entrenamiento. Dado que los archivos de entrenamiento pueden ser muy grandes, se utilizará el protocolo NFS v4 para acceder a una localización en la LAN donde se van a almacenar los archivos. En la aplicación configurada para un usuario, estos archivos están en la PC que tiene el cliente **Evolution**.

E.7 Mejoras a la interfaz de usuario

Las mejoras en la interfaz de usuario son posibles, si nos planteamos el diseño en una red corporativa, donde las tareas de codificación y entrenamiento pueden estar centralizadas en un equipo administrador de la aplicación. En ese escenario, los clientes configurarían su filtro anti*spam* con el grado de detección que quisieran tener; dentro del rango de Score de Spam de 0 a 100.

Si quisieramos mejorar la aplicación a nivel de un usuario, podrían implementarse controles nuevos al **Evolution**, que eviten la parte de codificación por línea de comandos, y de esa forma sería más transparente el procedimiento para el usuario.

Otra posible mejora sería la inclusión de un *tag* en el *Subject* informando si un mail es *spam*, para que luego utilizando ese *tag*, otra regla los clasifique convenientemente.

E.8 Integración a otras herramientas de correo

La integración con otras herramientas de correo es totalmente posible (Fig. 17) desde la interfaz de usuario de **Evolution**. El programa permite configurar un regla a continuación de otra (Fig. 17) definiendo para cada una de ellas, el umbral de detección de *spam*.

Los mensajes entrantes son procesados por cada regla, una a continuación de otra. El primero en detectarlo detiene el proceso.

Figura 17. Integración con otras herramientas de *spam* en *Evolution*

Herramienta Entrenamiento *spam*	Base	Cantidad de *mails*	Tiempo de entrenamiento (seg)
sa-learn	*spam*	1000	57
	ham	1000	99
	ham / spam	2000	2
trainsom -ds	*spam*	1000	10
trainsom -dh	*ham*	1000	16
trainsom -c	*ham / spam*	2000	1.4
som (PC)	*ham / spam*	2000	6
som (DSP)	*ham / spam*	2000	15

TABLE VI
TIEMPOS DE ENTRENAMIENTO *spamassasin* VS *somasspam*

VIII. COMPARACIÓN CON OTRAS HERRAMIENTAS ANTI *spam*

A. Introducción

El sistema de detección de *spam* que utilizaremos para comparar la efectividad, desempeño y manejo de nuestra aplicación, es el conocido filtro anti *spam* ***spamassassin*** de Apache Software [30]. Coincidentemente poseé funcionalidades muy parecidas a las desarrolladas por nosotros desde el punto de vista del manejo y configuración del sistema, por lo cuál resultará más simple evaluar la comparación. Veremos en la respuesta de ambos sistemas, para la detección de *spam* de un cliente de Correo Electrónico típico.

B. Comparación de tiempos de entrenamiento

Spamassassin requiere también de una etapa de entrenamiento, en éste caso aplica el algoritmo de Naïve Bayes. Para ello, fue necesario separar de nuestra base de *mails* de *ham* y *spam*, la cantidad de 1000 mensajes de cada tipo, siendo esta la cifra de entrenamiento óptima recomendada para cada cliente de la herramienta.

Un primer testeo fue el de los tiempos de codificación y entrenamiento de los sistemas (tabla VI.

Herramienta Anti *spam*	Base	Cantidad de *mails*	Clasificados Correctamente	Falsos Positivos	Porcentaje Eficacia	Demora Procesamiento
spamassassin	*spam*	500	442	58	88.4 %	24.5 *mins*
	ham	500	498	2	99.6 %	27 *mins*
somasspam	*spam*	500	475	25	95 %	41 *seg*
DSP	*ham*	500	465	35	93 %	35 *seg*
somasspam	*spam*	500	485	15	97.0 %	16.5 *seg*
PC	*ham*	500	407	93	81.4 %	11 *seg*

TABLE VII
COMPARACIÓN DE DESEMPEÑO Y EFICACIA *spamassassin* VS *somasspam*

C. Comparación del desempeño y de la eficacia en la detección

Observaremos cuál es el grado de detección en uno y otro sistema y los tiempos de aplicación del filtro para cada mensaje. Tomando una base de 500 mensajes nuevos de cada tipo, *spam* y *ham*, llegamos a la tabla VII donde esta expresada la respuesta en detección.

D. Comparación en el manejo de la aplicación

1. *spamassassin*

Como fue mencionado anteriormente, *spamassasin* requiere una etapa previa de clasificación y entrenamieto de sus bases para la posterior utilización del filtro.

Los comandos utilizados para ese fin son:

```
sa-learn --spam -C /etc/mail/spamassassin --showdots --mbox  dbspam
```

Para el aprendizaje de *spam*

```
sa-learn --ham -C /etc/mail/spamassassin --showdots --mbox  dbham
```

Para el aprendizaje de *ham*

Siendo *dbspam* [*dbham*] la base con los 1000 *mails* de *spam* [*ham*] necesaria para el entrenamiento.

El parámetro *mbox* es para identificar el formato del archivo de *mails* utilizado por **Evolution**.

Luego se ejecuta un comando de sincronismo entre las dos bases.

```
sa-learn -C /etc/mail/spamassassin --showdots --sync
```

Quedando pronta para ser utilizada

La etapa final consiste en configurar el **Evolution** para que filtre utilizando *spamassassin*. Los parámetros aplicables a este caso son :

```
/usr/bin/spamassassin -P -e
```

La figura 18 muestra la configuración.

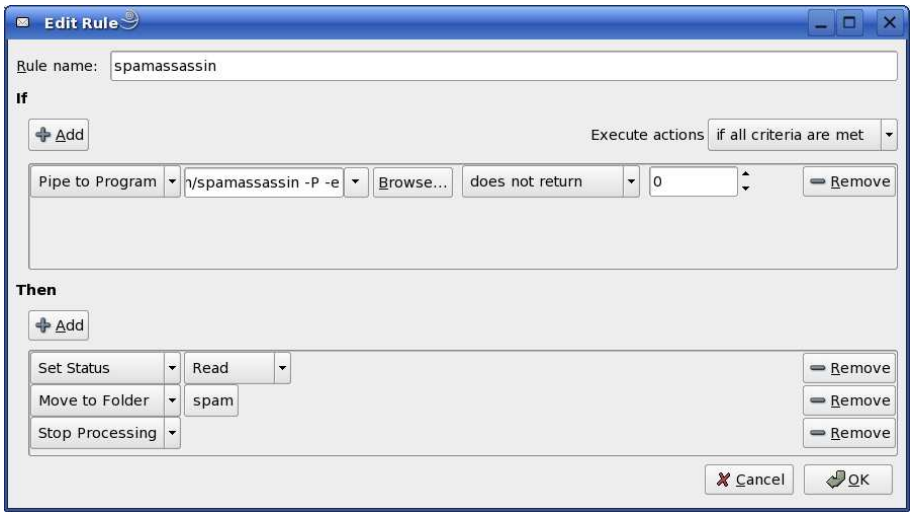

Figura 18. Configuración de spamassassin en *Evolution*

2. ***somasspam*** Nuestra aplicacion realiza la etapa de entrenamiento del lado del cliente, inicialmente sin necesidad de acceder directamente a los archivo *mbox* del usuario. Debemos configurar el filtro de **Evolution** para que utilice el programa *trainsom -dh* para crear las bases de *ham* y *trainsom -ds* para el *spam*, aplicados a sus respectivas carpetas, sin realizar ninguna otra acción sobre los mails allí guardados.

```
/etc/mail/somasspam/trainsom -dh [-ds]
```

La figura 19 muestra la configuración. Seleccionamos el conjunto de mails que querramos usar para el entrenamiento, como muestra la figura 20, y si quisieramos todos, se seleccionan con CTRL + A.

Y aplicamos el filtro desde el menú *Message / Apply Filters* o con CTRL + Y.

Luego será necesario desde el *shell* correr el comando de codificación de los mensajes

```
/etc/mail/somasspam/trainsom -c
```

Y finalmente si el *Servidor SOM* está activo ejecutamos

```
/etc/mail/somasspam/somasspam -t
```

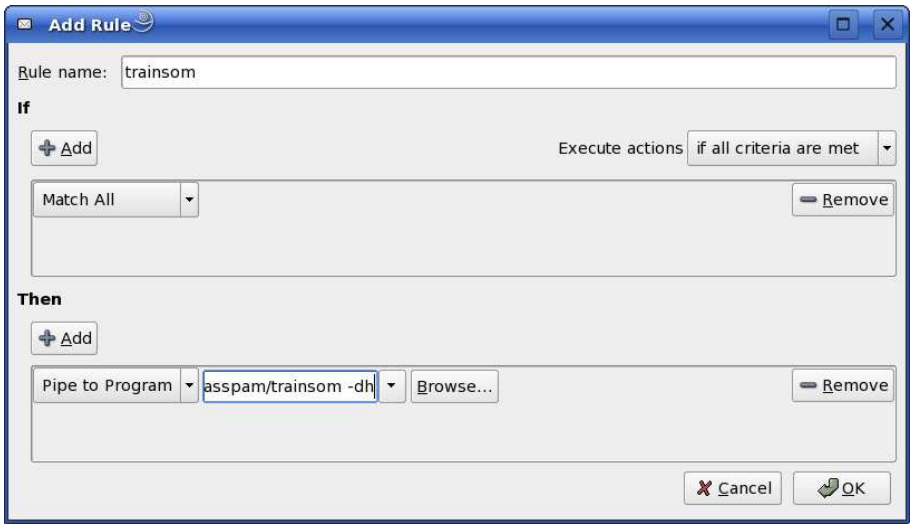

Figura 19. Configuración de *trainsom -dh (somasspam)* en *Evolution*

En caso contrario se debe iniciar la aplicación *som* del lado del servidor configurando el archivo *som.config* para que inicie entrenando el mapa.

El filtro de clasificación propiamente dicho, *somasspam -s* se aplica de identica manera que *spamassassin -P -e*.

En el caso de *spamassassin*, es necesario configurar el archivo

```
/home/username/.spamassassin/user_prefs
```

agregando la línea

```
required_score    nn.n
```

para que el *score* de *spam* tenga como umbral *nn.n* para detectar el spam. Este valor se logra a medida que se suman *hits* en cada regla configurada en el sistema [30].

En nuestra aplicación *somasspam*, tenemos la opción en la interfaz de usuario de ajustar el umbral de detección para la clasificación de un mensaje como *spam*, en la figura anterior, si el *Score de Spam* es mayor que 50 el Evolution mueve el mensaje a la carpeta **spam**, si es menor no efectúa ninguna otra acción, el mensaje queda disponible para el usuario, o para

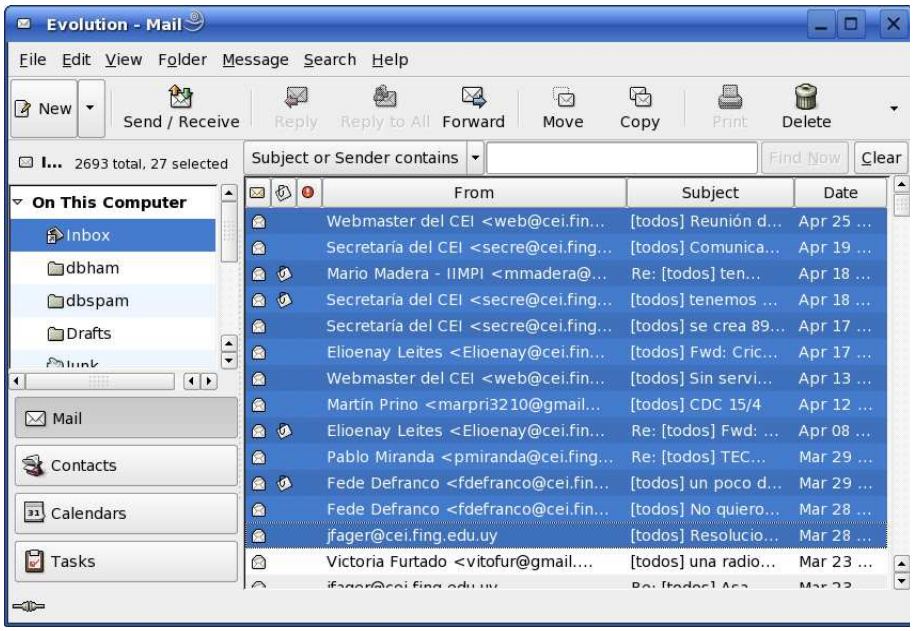

Figura 20. Selección de mails en *Evolution*

ser manejado por alguna otra regla.

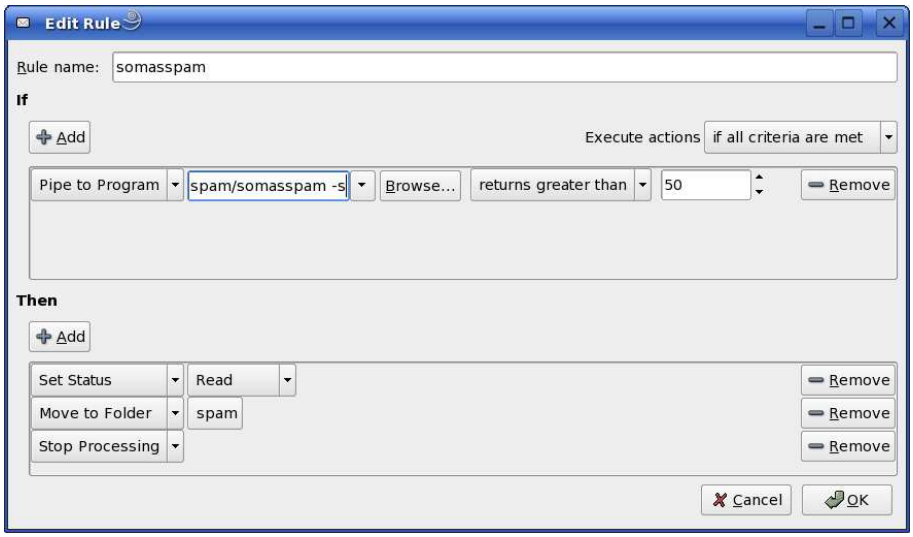

Figura 21. Configuración de *somasspam* en *Evolution*

IX. Conclusiones

A. Introducción

Se detallarán aquí las conclusiones obtenidas del Proyecto y posibles líneas de investigación futura.

B. Conclusiones generales

La implementación del modelo de red neuronal *SOM* en un hardware nuevo, la plataforma *DSP Blackfin*, ha sido plenamente satisfactoria ya que para un problema de test se obtienen los mismos resultados en la plataforma *Blackfin* y en un PC.

Las rutinas desarrolladas en *Assembler* han contribuido a mejorar el desempeño, aunque con un efecto menor al esperado. Una posible causa de este hecho es que en las versiones de *uClinux* manejadas durante el desarrollo del proyecto no estaba operativo el acceso a memoria eficiente a través del caché, para aplicaciones en C, lo cual influye en la performance dado que una parte importante de la aplicación SOM está programada en C. Muy recientemente (principios de Mayo de 2007) ha aparecido una nueva versión de *uClinux* con soporte para caché en C, pero únicamente para interfaz binaria de aplicación del tipo *FDPIC ELF*. Es importante tener en cuenta que el cambio de interfaz binaria de aplicación implica una reescritura de todas las rutinas de *Assembler* y gran parte de las rutinas de C.

Los resultados obtenidos en la fase de entrenamiento y detección de SPAM son buenos, comparables a herramientas comerciales de uso en el ambiente profesional. En concreto, respecto al spamassasin, estándar de facto en el campo *antispam*, se han obtenido unos resultados muy promisirios, aunque parece difícil llegar al nivel de falsos positivos proporcionados por spamassasin.

La solución parece tener buena escalabilidad, pero no ha sido probada en profundidad

la respuesta ante un incremento grande de las bases de correo.

La experiencia-usuario que tiene lugar en la operación de la herramienta desarrollada es esencialmente igual a la existente en la utilización de otras herramientas antispam comerciales. Este hecho facilita su despliegue en un ambiente corporativo.

Se puede utilizar la aplicación desarrollada como herramienta antispam de ambiente corporativo. Es muy interesante desde el punto de vista económico la implementación en la plataforma, dado que el precio de la placa de desarrollo ronda los USD 200, bastante menor que el precio de un servidor de mediano porte dedicado a esta labor.

A través del desarrollo del Proyecto se destacan los siguientes aspectos formativos alcanzados:

- Conocimiento del estado del arte en sistemas embebidos. La búsqueda de la plataforma para *SOM* concluyó en un acercamiento a un cierto número de las posibles arquitecturas de plataformas para sistemas embebidos, proporcionando un amplio panorama de la oferta mundial de procesadores en este campo.
- Desarrollo *opensource*. El presupuesto acotado condujo a prestar atención a las soluciones de desarrollo en software libre de patentes, en particular *Linux* en sistemas embebidos. La experiencia adquirida se aplica también a desarrollo en *Linux* en plataformas PC estándar dado que ambos procesos no difieren mayormente.
- Técnicas avanzadas de desarrollo de software. El proceso de desarrollo permitió adquirir destreza en técnicas nuevas para el grupo de proyecto, poco exploradas en otras asignaturas de la curricula ingenieril, como por ejemplo: compilación cruzada, *debugging* basado en *gdb*, *profiling*, compilación de *kernel*, etc.
- *Machine learning*. El estudio en redes neuronales permitió el acercamiento a técnicas de punta en aprendizaje automático.

- Antispam. Para la aplicación realizada fue necesario adquirir conocimiento sobre la configuración y utilización de las soluciones antispam existentes, de gran valor para administradores de redes

C. Líneas de trabajo futuro

La solución desarrollada tiene carácter de prototipo. Como tal se pueden introducir las siguientes mejoras:

1. Implementar mejoras en la codificación de mensajes. Profundizando en otras técnicas de codificación se pueden incrementar las prestaciones de la solución. Por ejemplo, se podría explorar agregar características dependientes del cuerpo del mail.
2. Nuevos algoritmos de clasificación. La exploración de otros sistemas de clasificación supervisada, además de la detección por votación, sobre SOM pueden mejorar las prestaciones. Una posibilidad es implementar *LVQ (Learning Vector Quantization).*
3. Automatización de los procesos antispam en el **Evolution**. Se pueden crear nuevas reglas en el **Evolution** de modo tal que la operación del antispam sea totalmente automática, en particular el reentrenamiento del sistema sea totalmente automático considerando el aumento del volúmen de *SPAM* recibido.
4. La herramienta se desarrollo originalmente para el cliente de email **Evolution**. Se puede adaptar esta herramienta para los otros clientes de correo, en particular para **Outlook**.
5. El conocimiento adquirido en el desarrollo en *uClinux* es un buen punto de partida para que, a través de un estudio del *kernel* de *Linux*, poder optimizar el mismo para un mejor desempeño de la *SOM.*
6. La performance de la red *SOM* en *Blackfin* se puede mejorar integrando al kernel de uClinux las funciones especificas de SOM, lo cual asegura un aumento del desempeño,

al correr en modo protegido.

7. Considerando el carácter "*opensource*" de la placa de desarrollo **STAMP** (se disponen de los esquemáticos completos) y el reducido costo del procesador **Blackfin bf537** (USD 5) es posible diseñar una versión reducida de la misma que abata los costos del prototipo original. Se estima que la nueva placa *Blackfin* tendrá un costo del orden de USD 50.

Referencias

[1] Teuvo Kohonen, *Self-Organizing Maps*, Springer, tercera edition, 2001.

[2] Teuvo Kohonen, *Self-organized formation of topologically correct feature maps*, Biological Cybernetics, vol.43 edition, 1982.

[3] Guido Deboeck and Teuvo Kohonen, *Visual Explorations in Finance with Self-Organizing Maps*, Springer, 1999.

[4] Teuvo Kohonen, Jussi Hynninen, Jari Kangas, and Jorma Laaksonen, *SOM_PAK: The Self-Organizing Map Program Package*, Laboratory of Computer, Helsinki University of Technology, 1999.

[5] Helsinki University of Technology Laboratory of Computer, *SOM Toolbox*, [en línea], $< http://www.cis.hut.fi/projects/somtoolbox >$, [consulta: 15 mayo 2007].

[6] Eudaptics gmbh, *Viscovery - SOMine*, [en línea], $< http://www.eudaptics.com/ >$, [consulta: 15 mayo 2007].

[7] Nenet development team, *Neural Networks Tool - Nenet*, [en línea], $< http://koti.mbnet.fi/ phodju/nenet/Nenet/General.html >$, [consulta: 15 mayo 2007].

[8] Wikipedia en español, *Definicion de spam*, [en línea], $< http://es.wikipedia.org/wiki/Spam >$, [consulta: 15 mayo 2007].

[9] iCAUCE.Ar, *Definicion de spam*, [en línea], $< http://wiki.cauce.org.ar/cgi-bin/moin.cgi/DefinicionDeSpam >$, [consulta: 15 mayo 2007].

[10] Robert Neumayer, *Ensemble Classification for Spam Filtering Based on Clustering of Text Corpora*, 2003.

[11] Wikipedia en inglés, *E-mail Spam*, [en línea], $< http://en.wikipedia.org/wiki/E-mail_spam >$, [consulta: 15 mayo 2007].

[12] Univesity of Bangkok, *Anti-Spam Filtering*, KASPERSKY Lab, 2005.

[13] EDN staff, *Electronics Design News Magazine*, [en línea], < *http* : *//www.edn.com* >, [consulta: 15 mayo 2007].

[14] Blackfin uClinux proyect team, *Linux on the Blackfin Processor*, [en línea], < *http* : *//blackfin.uclinux.org/gf/* >, [consulta: 15 mayo 2007].

[15] Karim Yaghmour, *Building Embedded Linux Systems*, O'Reilly, tercera edition, 2003.

[16] Eclipse Foundation, *Eclipse - an open development platform*, [en línea], < *http* : *//www.eclipse.org* >, [consulta: 15 mayo 2007].

[17] CDT, *CDT Eclipse plugin para C/C++*, [en línea], < *http* : *//download.eclipse.org/tools/cdt/releases/eclipse*3,1 >, [consulta: 15 mayo 2007].

[18] GNU Proyect, *Data Display Debbuger*, [en línea], < *http* : *//www.gnu.org/software/ddd* >, [consulta: 15 mayo 2007].

[19] Hand Held Museum, *MB Simon series*, [en línea], < *http* : *//www.handheldmuseum.com/MB/Simon.htm* >, [consulta: 15 mayo 2007].

[20] The Open Group, *The Single UNIX Specification, Version 2*, [en línea], < *http* : *//opengroup.org/onlinepubs/*007908799/*xsh/pthread.h.html* >, [consulta: 15 mayo 2007].

[21] Simon Haykin, *Neural Networks - A Comprehensive Foundation*, Pearson Prentice Hall, segunda edition, 1999.

[22] School of Information and University of California Computer Science, *Machine Learning Database*, [en línea], < *http* : *//ftp.ics.uci.edu/pub/mlearn/databases* >, [consulta: 15 mayo 2007].

[23] Oprofile development team, *Oprofile - A System Profiler for Linux*, [en línea], < *http* : *//oprofile.sourceforge.net* >, [consulta: 15 mayo 2007].

[24] John Hennesy and David Patterson, *Computer Architecture: A Quantitative Approach*,

Morgan Kaufmann Publishers, tercera edition, 2003.

[25] Groupware Suite, *Evolution 2.4.0*, Novell, Inc. and Others, 1999-2005.

[26] Michael Zucci, *Evolution Plugin Development Manual*, Novell Inc., 2006.

[27] Enrico Blanzieri and Anton Bryl, *A survey of anti-spam techniques*, University of Trento, September, 2006.

[28] The SPF Proyect, *Sender Policy Framework*, [en línea], $<$ *http* : $//openspf.org/faq.html >$, May, 2006.

[29] Harris Drucker and VladimirÑ. Vapnik, *Support Vector Machines for Spam Categorization*, IEEE Transactions on Neural Networks, 1999.

[30] Apache Software, *The SpamAssassin(tm) Project*, [en línea], $<$ *http* : $//spamassassin.apache.org >$, 2007[consulta: 15 mayo 2007].

[31] Analog Devices Inc., *ADSP-BF53x/BF56x Blackfin Processor Programming Reference*, Analog Devices, 2006.

[32] Analog Devices Inc., *ADSP-BF537 Blackfin Processor Hardware Reference*, Analog Devices, 2005.

[33] Analog Devices Inc., *Analog Devices*, [en línea], $<$ *http* : $//www.analog.com >$, [consulta: 15 mayo 2007].

[34] Intel Inc., *Intel*, [en línea], $<$ *http* : $//www.intel.com >$, [consulta: 15 mayo 2007].

Appendices

A. Apéndice: Descripción del procesador Blackfin

A. Introducción

Se describirán a continuación las principales caracteristicas del procesador Blackfin. La información relevante para este apéndice ha sido adaptada de [31], [32], [33], [14] y [34]

B. Micro Signal Architecture

Analog Devices e **Intel** desarrollaron conjuntamente el núcleo de procesador denominado *Micro Signal Architecture (MSA)* a finales de 2000. Su objetivo de diseño fue combinar las capacidades de una arquitectura DSP con funcionalidad integrada de microcontrolador en el marco de las aplicaciones de telecomunicaciones en dispositivos accionados por baterías.

Se lograron mejoras de performance, consumo y un modelo simplificado de programación La idea general era integrar el procesador DSP dedicado a audio-video y el microcontrolador dedicado a tareas de control en un solo *core*.

Características DSP:

- 1 bus de instrucciones.
- 2 buses separados de datos sobre un único espacio de direcciones de 4Gb.
- 2 multiplicadores de 16 bits de ciclo único.
- 2 ALUs de 40 bits, las cuales pueden operar en modo 8,16,32 o 40 bits. En un solo ciclo de instrucción se pueden efectuar dos operaciones de 32 bits o cuatro de 16 bits o cuatro de 8 bits.
- 1 barrel shifter de 40 bits.
- 2 punteros de 32 bits para las ALUs con soporte para direccionamiento circular y de bit invertido.
- 2 contadores de loop hardware de 0 overhead.

- Soporte hardware para saturación y clipping.
- 4 ALUs de 8 bits auxiliares para realizar operaciones de procesamiento de imagenes.

Características *µcontrolador*:

- Manipulación arbitraria de bits.
- Codificación de instrucciones mixta en 16 y 32 bits, para una mayor densidad de codigo.
- Protección de memoria.
- Punteros al Stack y memoria especial SDRAM para hacer cambios de contexto.
- Control de potencia consumida flexible.
- Mecanismo de interrupciones extensible, anidado y priorizado para tiempo real.

Características de programación:

- Arquitéctura optimizada para en C/C++, lo cual hace que en aplicaciones comunes mas del 80 % del código se pueda mantener como C/C++.
- Unidad de debugging con de 6 direcciones y 2 breakpoints.
- Buffer de trazado con 16 entradas.

Intel utilizó la nueva arquitectura en chipsets de teléfonos celulares de última generación.

Analog Devices construyó la línea de procesadores **Blackfin** en base al nuevo *core*.

C. Familia Blackfin

El Blackfin es un procesador de alta performance de 16/32 bits con un pipeline de 10 etapas, set de instrucciones de longitud variable, hibrido microcontrolador-DSP y con soporte SIMD[14], es decir para instrucciones que actuan sobre varios datos a la vez.

El costo de un procesador Blackfin bf537 es de aproximadamente USD 5, en grandes cantidades.

[14] Single Instruction Multiple Data

Periféricos

Contiene soporte onchip para los siguientes periféricos:

- Controlador DMA.
- Ethernet 10/100.
- CAN. Es un bus industrial de transmisión de datos, que se usa en la industría automotriz.
- UART.
- Temporizadores con soporte PWM.
- Reloj de tiempo real.
- SPI. Interfaz de periférico serial.
- PPI. Interfaz de periférico paralelo.
- Watchdog. Subsistema que resetea a todo el sistema cuando se cumplen determinadas condiciones.
- GPIO. Banderas programables de proposito general para entrada/salida.

Estructura interna

La estructura interna del Blackfin esta decripta en la figura 22

Los bloques tienen las siguientes características:

Registros de proposito general Existen 8 registros de proposito general (R0-R7) de 32 bits los cuales puden ser utilizados con los siguientes tipos de datos:

- enteros de 8, 16 o 32 bits con y sin signo
- números fraccionarios punto fijo de 16 o 32 bits con o sin signo

Se soportan dos lecturas de 32 bits y dos escrituras de 32 bits en un solo ciclo de reloj. 6 registros punteros de dirección (P0-P5) de proposito general de 32 bits. Un puntero de stack SP. Un puntero de frame FP utilizado para cambios de contexto en procesos.

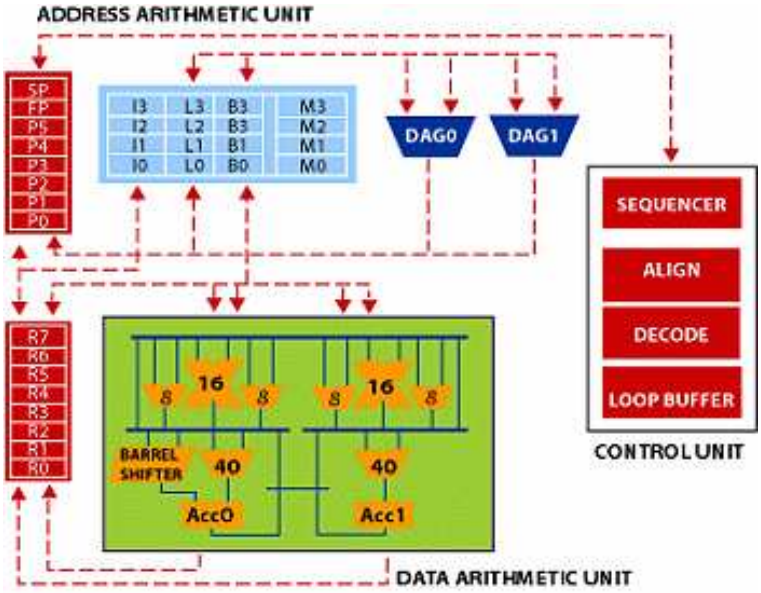

Figura 22. Estructura Interna del procesador *Blackfin*

Unidad aritmética de datos Esta formada por 2 MACs de 16 bits, 2 ALUs de 40 bits, 4 ALUs de 8 bits y un barrel shifter de 40 bits. Los dos acumuladores de 40 bits pueden funcionar con distintos modos de acumulación. Cualquiera de los registros de proposito general se pueden utilizar para calculos como operandos de 8, 16 o 32 bits. Se puede acceder a las mitades de 16 bits de los registros de forma arbitraría.

Unidad aritmética de direcciones Existen dos generadores de direcciones de datos (DAG) los cuales pueden realizar dos accesos simultaneos a la memoria Los DAG utilizan los siguientes grupos de registros de 32 bits:

- Indice (I0-I3)
- Longitud (L0-L3)
- Modificación (M0-M3)
- Base (B0-B3).

Estos grupos de registros se utilizan para direccionamiento circular. También se puede direccionar con los registros punteros de direcciones P0-P5, de forma similar al C.

Otros modos de direccionamiento soportado son indirecto, indexado, autoincremento, autodecremento y bit reverso.

Unidad secuenciadora de programa Es la unidad que controla el flujo de programa e implementa los saltos condicionales y los llamados a subrutinas. La electrónica interna del procesador se encarga de asegurar en todo momento que no existan problemas de instrucciones con datos corruptos por el pipeline, retrasando la ejecución cuando es necesario, de forma totalmente transparente para el programador. Se soportan instrucciones de 16 y 32 bits de longitud, así algunas instrucciones limitadas de 64 bits.

Organización de la memoria

En la figura 23 se muestra un esquema de la memoria del **Blackfin bf537**. El espacio total de memoria es de 4G bytes, accesible por direcciones de 32 bits. Aquí se mapea todo tipo de memoria y también los puertos de entrada salida. Se distinguen:

- Memoria interna. Existen tres tipos
  - L1 de instrucción. Está formada por memoria SRAM que se puede configurar como un cache de 4 vías para instrucciones.
  - L1 de datos. Es memoria SRAM configurable como caché de dos vías para datos.
  - L1 scratchpad RAM. Esta memoria no puede ser configurada como caché, y por lo general se usa para cambios de contexto rápidos.
- Memoria externa. El blackfin contiene un controlador de SDRAM para 512MB y un controlador de memoria asíncrona que soporta 4 bancos de 1MB cada uno.
- Dispositivos de Entrada/Salida. No hay un espacio separado de entrada/salida. Los periféricos onchip se configuran y controlan con registros mapeados en memoria (MMR).

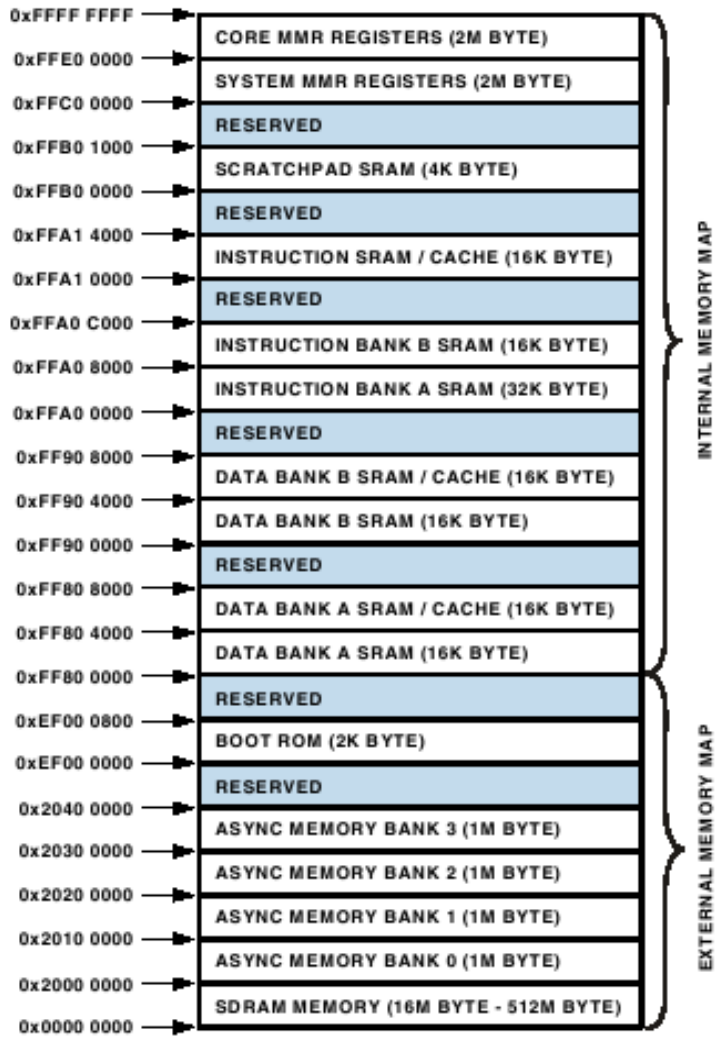

Figura 23. Mapa de memoria del procesador *bf537*

El acceso a la memoria se debe realizar *alineado*, es decir si se quiere leer una palabra de 32 bits el puntero utilizado debe apuntar a una dirección de memoria que sea múltiplo de 4. De la misma forma un acceso de 16 bits ser realizado por un puntero apuntando a una dirección par.

El **Blackfin** dispone de una unidad de manejo de memoria basica que implementa acceso protegido a memoria modo supervisor y modo usuario, la cual facilita la implementación de un sistema operativo.

Manejo de potencia dinámico

El **Blackfin** contiene un mecanismo hardware que selectivamente apaga las unidades funcionales si no registran actividad. Soporta diversos modos de operación con potencia reducida para periodos de poca actividad de CPU. El sistema de control de potencia actua sobre la frecuencia y el voltaje del procesador. Todo esto es controlado por el sistema operativo.

B. Apéndice: Contenido del CD adjunto

A. Introducción

Los archivos relevantes del proyecto se incluyen en CD adjunto. Se puede acceder a ellos a través de cualquier computadora que disponga de una unidad de *CDROM* capaz de leer un sistema de archivo *ISO/JOLIET*. No es necesario ningún software especial para acceder a los mismos, salvo un lector de archivos *PDF*, para los cuales se recomienda cualquier versión opensource, por ejemplo el *KPDF*.

B. Estructura de directorios

El CD se divide en dos directorios principales:

1. **Blackfin** Se incluye información y herramientas de desarrollo para la plataforma STAMP BF537.
   a) *documentacion* Está organizado de la siguiente forma:
      1) *hardware* Documentación de hardware de la familia de procesadores *Blackfin*.
      2) *software* Documentación de software sobre la familia de procesadores *Blackfin*.
      3) *uClinux* Documentación general acerca de *uClinux* en la placa *STAMP*.
   b) *herramientas*
      1) *toolchain* Versión para *Blackfin* de la *gnu toolchain*.
      2) *uboot* Booteador universal para plataforma *Blackfin*.
      3) *uClinux* Fuentes del *kernel* de *uClinux* para plataforma *Blackfin*.
2. **proyecto** Este directorio contiene la documentación del proyecto y los siguientes subdiretorios:

a) *imagen* Incluye el *kernel* que necesita correr la plataforma *Blackfin* para la aplicación *antispam*.

b) *somasspam* Programas de la aplicación antispam. Contiene las fuentes y los ejecutables.

c) *mas programas* Contiene otros programas de demostración. Presenta el siguiente esquema de subdirectorios.

1) *perceptron* Son las fuentes y los ejecutables de una aplicación que emula la red perceptron utilizando las librerías *pthreads*.

2) *simon* Implementación en la placa *STAMP BF537* del popular juego *simon*.