ESCOLA TÈCNICA SUPERIOR ENGINYERIA
UNIVERSITAT ROVIRA I VIRGILI
Departament d'Enginyeria Electrònica Elèctrica i Automàtica

Traducción automatizada basada en algoritmos de redes neuronales artificiales

Titulación: Enginyeria Tècnica Industrial en Electrònica Industrial

AUTOR: Xavier Escobar Sola
DIRECTORES: Jesús Brezmes, Eduard Llobet

Fecha: Junio / 2009 .

Dedico todo el esfuerzo aquí dedicado a la entera humanidad, una raza con mucho camino realizado lleno de esfuerzos, y anhelando que lo más bonito sea el porvenir, un mundo para compartir, mejorar y explorar.

Gracias a mis profesores Brezmes y Llovet por dar lo mejor de ellos mismos y transmitir su conocimiento y sobretodo a mi madre Alicia, mi hermano Dani, mi novia Laia y amigos por aguantar a esta cabeza loca.

Traducción automatizada basada en algoritmos de redes neuronales artificiales

ÍNDICE GENERAL

Titulación: Enginyeria Tècnica Industrial en Electrònica Industrial

AUTOR: Xavier Escobar
DIRECTORES: Jesús Brezmes, Eduard Llobet

Fecha: Junio / 2009 .

1.- Indice general

Traducción automatizada basada en algoritmos de redes neuronales artificiales

MEMORIA

Titulación: Enginyeria Tècnica Industrial en Electrònica Industrial

AUTOR: Xavier Escobar
DIRECTORES: Jesús Brezmes, Eduard Llobet

Fecha: Junio / 2009 .

2.- Memoria

Índice de la memoria:

2.1- Objeto

El aprendizaje automático puede entenderse de forma general como el proceso mediante el cual se consigue “Sintetizar procedimientos capaces de realizar tareas de las que sólo tenemos algunas descripciones parciales sobre el modo en que nos gustaría que se llevasen a cabo”. [1]

Con este marco se presenta el siguiente proyecto, básicamente basado en software, que tiene como finalidad el desarrollo de todas las funciones necesarias para poder realizar el entrenamiento de elementos lingüísticos en redes neuronales artificiales, explorando por el camino diferentes opciones existentes, escogiendo en cada momento el mejor método para poder lograr redes neuronales con mayor capacidad de aprendizaje.

La red utilizada ha sido la perceptrón multicapa, utilizando la de mayor grado de generalización conocida: la BackPropagation.

El estudio se ha realizado con una aplicación de traducción a pesar de poderse aplicar a posteriori a otros tipos de entrenamiento como los silogismos, conversaciones humanas, matemáticas o cualquier otra ciencia, dado el grado de generalización de los algoritmos utilizados.

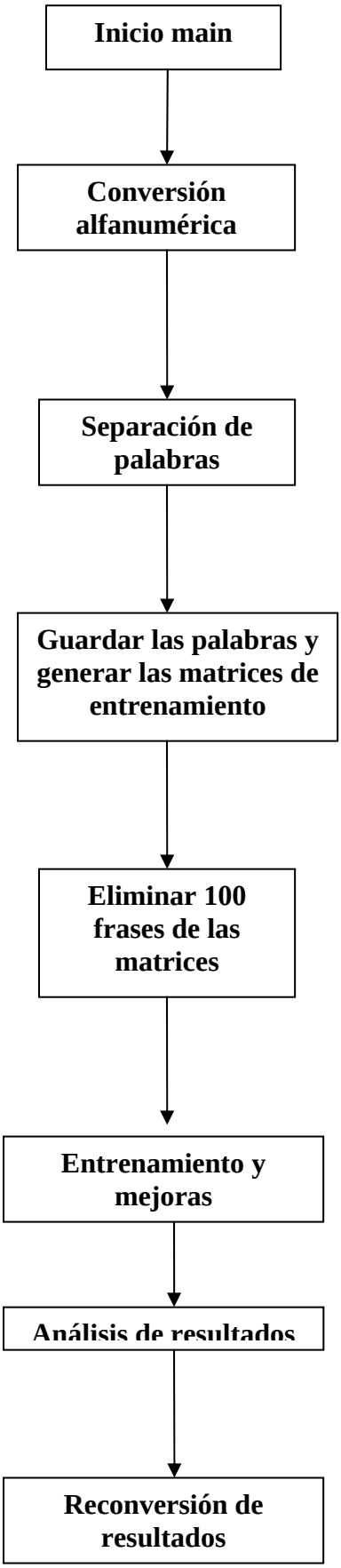
Dada esta premisa, podemos intuir que ha de ser posible crear una herramienta algorítmica que nos permita obtener conclusiones lógicas dado un determinado entrenamiento y un texto de entrada apropiado, y es el estudio de este conjunto de algoritmos en el que se basa este trabajo.

Se plantea básicamente como un estudio de viabilidad, para un proyecto futuro más complejo, anexado a un sintetizador de voz y un reconocimiento de voz, análisis palabra a palabra, entre otras mejoras. Pero para realizar este análisis ha sido necesario crear todo un programa que sea capaz de realizar unas pruebas más o menos complejas y analice los errores.

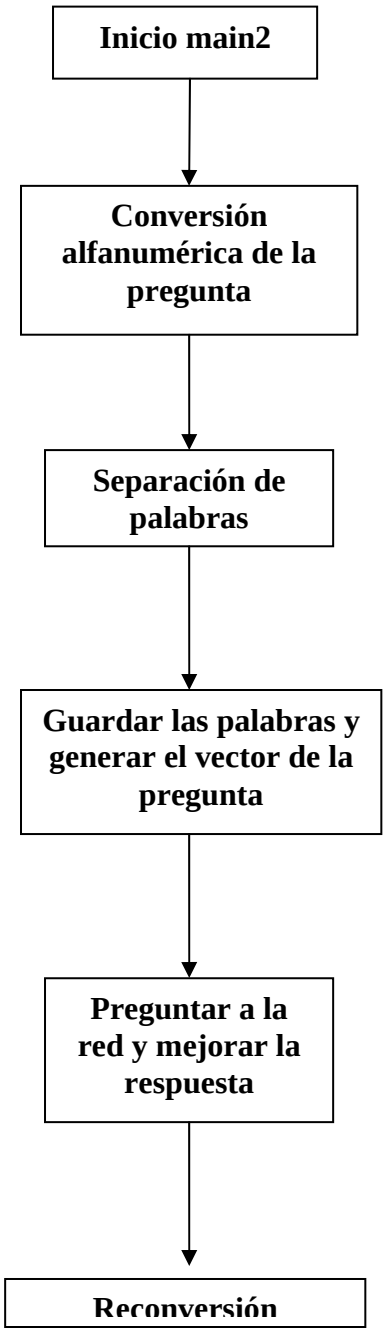
Se espera poder conseguir unas traducciones satisfactorias, dando a conocer solo una descripción parcial de ellas, es decir, de todas las traducciones de frases que nos interesa que logre contestar, solo se le mostrará una parte en el entrenamiento.

A continuación se presentan los diagramas de bloques generales de los dos programas.

2.1.1- Programa 1



2.1.2- Programa 2



2.2- Alcance

En este proyecto se esperan alcanzar dos programas de traducción, similares entre ellos, pero con diferentes propósitos.

1.- El primero de ellos está encaminado a la investigación y entrenamiento mediante redes neuronales artificiales (RNA's). Esto significa que en este programa se puede obtener una RNA entrenada, a partir de un texto de entrada con un cierto número de ellas. Una vez tengamos esta red entrenada, se procede a utilizarla en el segundo programa para realizar consultas específicas.

2.- El segundo programa está encarado ya hacia el usuario final, teniendo las RNA's ya entrenadas, se da la opción al usuario para solicitar una traducción cualquiera y servirá de base para futuros proyectos de éste ámbito. El uso de este programa requiere imperativamente poseer una red entrenada para realizar la consulta, pero posee una RNA que se carga por defecto, pero que el usuario puede intercambiarla con la suya propia, creada en el programa1.

Cada uno de los programas, tendrá tres partes diferenciadas: conversión de datos, entrenamiento de datos y reconversión de datos.

La aproximación escogida se basa en ir incrementando secuencialmente la complejidad del conjunto de entrenamiento, para ir identificando y corrigiendo fallos poco a poco. La meta para este proyecto está en analizar que grado de acierto se puede alcanzar realizando un entrenamiento con más de cincuenta palabras para cada uno de los dos idiomas, realizando un entrenamiento de 1900 frases y analizando los resultados en más de 2000 frases.

No pretende ser un programa definitivo, pero sí muy bien estructurado, con partes claramente diferenciadas y funciones potentes, con análisis de errores en cada función y con funciones altamente generalizadas pero a la vez sencillas y comprimidas; utilizando las mínimas variables intermedias posibles. Teniendo muy en cuenta, que no es conveniente tener cargadas muchas variables de matrices de miles de filas y columnas a la vez, ni tenerlas cargadas en momentos críticos, como durante el entrenamiento de la red, para evitar problemas comunes por falta de memoria RAM.

2.3- Antecedentes

Las redes neuronales han sido una herramienta con una aplicabilidad cada vez mayor en las últimas décadas. Siendo conocedores del potencial que conlleva utilizar una herramienta como las redes neuronales artificiales (RNA), con la cual, dado un conjunto par de datos entrada/salida de una determinada función (lineal o no), nos permite calcular de una manera fácil para el usuario, pero compleja matemáticamente, una respuesta lógica (si existe), dada una entrada no conocida previamente. Las RNA, son consideradas aproximadores universales, es decir, que tienen la capacidad de realizar una progresión cada vez mejor, dada una función no aleatoria, de variables tanto de entrada como de salida multivariantes.

Hay dos corrientes fundamentales en el aprendizaje automático: los sistemas que sintetizan reglas (u otros mecanismos equivalentes) capaces de clasificar casos no vistos en un conjunto finito de categorías discretas, y los que inducen funciones. En este proyecto estos dos planteamientos se combinan para obtener un nuevo método que aprende funciones usando, como herramienta fundamental, un sistema de aprendizaje automático discreto. El logro de este mecanismo es que maneja, de forma natural, tanto símbolos discretos como números en los valores de los atributos que definen los casos de entrenamiento y los no vistos.

La inspiración básica del proyecto ha venido del programa Dr. Abuse. Este programa de Inteligencia Artificial (AI) que demuestra apariencia humana, basado en el famoso programa [2]Eliza de Joseph Weizenbaum. Permite mantener una conversación divertida e inteligente con el ordenador. El programa ha sido entrenado en cientos de conversaciones con usuarios humanos o robóticos a través de Internet y otros medios, incorporando comandos específicos para dar órdenes como el de apertura de un programa determinado: explorador de Internet, reproductor de sonido, etc...
Dr. Abuse incluye, entre otras cosas, una potente base de datos de respuestas, una potente base de datos de temas que tratar y un módulo de comprensión de lenguaje natural, que me resultó de gran interés. Además del texto escrito, también habla en viva voz las frases. A pesar de ser en código cerrado, se conoce que tiene un entrenamiento en chats de Internet. Este programa es capaz de funcionar, definiendo frases como entrada y otras como salida, así dada una determinada frase (entrada), de una manera coherente, aprende su mecanismo de traducción, para poder en otros casos similares, resolver satisfactoriamente una nueva propuesta.

Mi proyecto se basa en descubrir una manera de realizar este proceso, pero aplicado inicialmente a la traducción de textos, aunque con posibilidad de aportar esta capacidad a otros campos de forma muy sencilla y basada no tanto por reglas, sino por experiencia en frases traducidas, lo que permite un fácil entrenamiento.

A parte de este proyecto, ha habido otros proyectos relacionados con el aprendizaje de lenguaje natural, y me interesa mencionar el “Cyc Project”, aunque todavía no posea ninguna referencia hacia la traducción de textos.

El proyecto Cyc comenzó el año 1984, destinado precisamente al entrenamiento de redes, organizado a partir de un conjunto de micro teorías para poder abarcar diferentes ramas del

conocimiento y tratando el lenguaje natural. Este proyecto presume de poseer la base de datos más grande del mundo, conteniendo acerca de 200.000 términos.
Esta aplicación intenta abarcar muy diferentes campos, como la búsqueda web avanzada, protecciones de seguridad informática, inteligencia robótica, realidad virtual, simulación de personajes inteligentes en videojuegos, y otras.

Así que tal vez, esta herramienta de traducción pueda ser algún día incorporada al proyecto Cyc, para completarlo con más aplicaciones.

El resto de programas de traducción (muy utilizados hoy en día) se basan en algoritmos muy complejos, programando cada una de las peculiaridades de la traducción natural, lo que las hace inmensamente problemáticas y dificultosas; a causa de este hecho, su tasa de éxito no es muy grande.

2.4- Normas y referencias

2.4.1- Disposiciones legales y normas aplicadas

Al utilizar el programa Matlab, se queda sujeto a las disposiciones legales que poseen:

http://www.mathworks.es/company/aboutus/policies_statements/

2.4.2- Bibliografía

La documentación proporcionada por los profesores, ha sido de gran utilidad, dos años atrás, cuando les notifiqué mi interés por el aprendizaje, me entregaron documentación muy útil. Además, me explicaron los fundamentos de las RNA y me pusieron a prueba con ejercicios simples para resolver. El documento más importante entregado ha sido: “Nnet”.

Páginas web:

http://es.wikipedia.org/wiki/Joseph_Weizenbaum

http://www.cyc.com/cyc/company/employment

http://www.psicoactiva.com/abuse/drabuse.htm

http://www.codeproject.com/

2.4.3- Programas de cálculo

El programa utilizado ha sido el Matlab, versión 7.0. El hecho de trabajar con Matlab no ha sido una consideración tomada a la ligera, pero se ha tenido en cuenta que posee funciones más potentes que “C” y otros compiladores similares, así como un entorno con gran facilidad para la manipulación de variables, especialmente matrices (transpuesta, obtención de número de filas/columnas, …) y su visualización. Si se deseara trabajar con interrupciones, sí que es posible que mereciese la pena compilarlo en “C”, realizando su conversión, pero de momento ésta posibilidad no se requiere, y en el futuro es posible que tampoco se necesite, debido a que se puede anexar a programas que creen ficheros para introducir los datos. Así que, a pesar de ser un programa de pago, sus ventajas compensarán este problema. Además Matab puede ser utilizado bajo diferentes sistemas operativos.

2.4.4- Plan de gestión de la calidad aplicado durante la redacción del Proyecto

Se ha comprobado la aplicabilidad del proyecto con 3 modelos de ordenador diferentes, con las siguientes características:

1.- Athlon a 1600 MHz con 1 GB de memoria RAM. Windows XP.
2.- Pentium IV a 1600 MHz 1 GB de memoria RAM. Windows Vista.
3.- Quad Core*2,6 GHz 2 GB de memoria RAM. Windows XP.

El hecho de trabajar con tres ordenadores da la facilidad de que mientras en uno se puede estar programando la siguiente versión del programa, en el otro se puede poner a prueba la anterior versión, para hacer diferentes análisis de estabilidad y eficacia del programa.
Se deben alternar entrenamientos largos, con cortos. Los primeros sirven para analizar el aprendizaje de la red, mientras que estos últimos servirán sobretodo para poner a prueba el programa, con entradas de datos extremos (pocas frases, mal construidas, con longitud variable,…) y analizar su correcto funcionamiento, procurando detectar anomalías mediante la visualización de las variables.

2.5- Definiciones y abreviaturas

Para el propósito de este proyecto se aplican las definiciones dadas en las Norma UNE-EN ISO 9000 junto con las siguientes (en el caso de que alguna de las siguientes esté en las normas anteriormente mencionadas, prevalece la definición dada en esta Norma):

RNA:

Abreviatura de la catalogación dada a una red neuronal artificial, con los algoritmos que conlleva.

Cross validation:

Para comprobar la efectividad de una RNA, teniendo un conjunto "N" de pares entrada/salida, se opta por entrenar "N-X" y posteriormente analizar el conjunto "N" total. En este caso se ha optado por dar a "X" el valor 100, que corresponde al 10% del conjunto "N".

Array:

Variable vector de [n×1] o [1×n] dimensiones.

Conversión alfanumérica:

Se realiza una conversión donde cada letra pasa a ser un número, codificado en ASCII [3].

Random:

Valor aleatorio generado por una función. En caso de ser un *array*, cada uno de los valores que lo integran debe ser un valor aleatorio generado por esa función.

Epochs:

Los epochs definen la cantidad de veces que se ha procedido o que se puede llegar a proceder a actualizar los pesos y bias de la RNA durante el proceso de entrenamiento. Un epoch es un ciclo completo de actualización en el que se utilizan la totalidad de datos del conjunto de entrenamiento.

Flag:

Variable auxiliar de apoyo, en nuestro caso binaria, que sirve de indicador, normalmente para conocer si el programa ha pasado por una parte del código en concreto.

Búffer:

Elemento capaz de almacenar información temporalmente, en este caso una variable que sirve para ir almacenando y recopilando datos hasta que llegue el momento de volcarlos a otro lugar.

Silogismo:

El silogismo es una forma de razonamiento deductivo que consta de dos proposiciones como premisas y otra como conclusión.

Ejemplo:

1ª premisa: Todos los hombres son mortales.
2ª premisa: Yo soy un hombre.
Conclusión: Por tanto, yo soy mortal.

2.6- Requisitos de utilización

Para garantizar el correcto funcionamiento del programa en cuestión, se requieren una serie de condiciones que garanticen su calidad y deben servir al usuario de guía.

El software ha sido diseñado para trabajar con Matlab, versión 7.0, bajo cualquier sistema operativo que lo acepte: Linux, Windows, Solaris y Macintosh.

En cuanto a software, hace falta tener colocados todos los ficheros de funciones dentro de una misma carpeta y seleccionarla como "current directory" en el Matlab. Y ejecutar "main" o "main2" según convenga.

Existe un entrenamiento por defecto, pero se puede crear uno propio. Todas las frases deben estar acabadas por un punto. Y entre palabra y palabra debe existir un espacio o una coma.

A pesar de que no es necesario que el texto de entrada sea en código ASCII, el punto, la coma y el espacio si que deberán serlo, debido a los condicionantes introducidos en el código "WordsSeparation".

Estos requisitos de utilización han sido el resultado de tener en cuenta los siguientes parámetros: facilidad para el usuario, efectividad superior de la RNA y finalmente facilidad para la programación.

La facilidad para el usuario, se garantizará mediante el uso del Matlab, así como la separación en los dos programas, que son ejecutados con una sencilla instrucción.

En cambio, para una mayor efectividad, es necesaria en esta versión la correcta configuración de "language":
Utilizando el texto de entrenamiento actual, si se trata de una traducción español→inglés (language=2) o inglés→español (language=1).

2.7- Análisis de soluciones

El programa consiste en una serie de scripts encadenados que van realizando los diferentes procesos y son llamados desde main o main2, según proceda entrenar la red o ponerla a prueba.

La primera cuestión que se planteó fue saber con qué tipo de datos iban a ser entrenadas las RNAs, qué forma debían tener las matrices y qué conversiones debían ser efectuadas.

Para ello, se generó un conjunto de pruebas entrenando elementos simples de palabras y frases, habiendo realizado una conversión alfanumérica letra a letra, teniendo tantos símbolos numéricos en el entrenamiento como letras y viendo unos resultados francamente pésimos, logrando traducir solo en un 50% de los casos palabras individuales, y con un 0% de éxito traduciendo frases con 4 palabras.

Otra opción planteada era simbolizar cada frase con un número, método que podría tener cierta validez a la hora de crear redes neuronales destinadas al diálogo, así ante una frase cualquiera de entrada, la red respondería con otra frase del conjunto de aprendizaje. Pero en el caso de la traducción no es muy recomendable, dado al gran número de frases de entrada que requeriría el aprendizaje de cualquier frase, la información sería muy limitada.

Una vez testeadas estas posibilidades, el siguiente paso lógico fue realizar un entrenamiento palabra a palabra, adjudicando un número a cada palabra, para así formar las matrices de entrada y salida, con un vector por cada frase.

Inicialmente, las palabras nuevas eran almacenadas en una sola matriz, lo que permitiría el entrenamiento en un área cualquiera, teniendo como condición una misma lengua tanto en la entrada como salida de la red. Para poder tener dos idiomas diferentes, la mejor solución que hallada ha sido crear dos archivos “words” y “words2”.Cada uno de estos archivos es utilizado para guardar todas las palabras conocidas de uno de los idiomas y así evitar resultados erróneos en ambos idiomas.

También ha sido implementada una función para seleccionar cual es el idioma de entrada y cual el de salida, es decir, si se trata de una traducción español→inglés o inglés→español. La variable que lo controla es “language” y actualmente puede tener dos posiciones: “1” o “2”. Actúa como una variable binaria, solo que está preparada para futuras actualizaciones, siendo una variable numérica.

Se ha separado el proyecto en dos programas diferenciados, el primero se llama con la función de “main” y es el encargado de ejecutar el entrenamiento de la red. En la variable “preload” se carga el archivo “in” y “language”.

El segundo programa es utilizado para preguntar frases individuales, teniendo la RNA entrenada previamente, bien sea por el usuario mediante el uso del primer programa o por defecto con una RNA ya entrenada. Este programa es ejecutado con la instrucción “main2”. La frase a preguntar debe ser colocada en la variable “ask”.

A la hora de obtener datos sobre la efectividad de una determinada RNA entrenada, se ha optado, por recomendación de los profesores, por una validación cruzada (*cross*

validation). Este proceso consiste en dejar 100 frases del conjunto total sin ser entrenadas, para el posterior análisis de todas las frases. Se debe realizar este proceso 10 veces consecutivas con 0 errores para considerar la red como validada. Para realizarla, ha sido necesario crear un *array random*, pero que debía tener la peculiaridad de no tener valores repetidos. Para ello se ha tenido que codificar una rutina que modificase el valor del número repetido. Para poder hacer un borrado secuencial de los archivos "finalsentenceReduced" y "finalsentenceOutReduced", también ha sido necesario ordenar esos números aleatorios en orden decreciente y así empezar el borrado por el final del fichero, hasta llegar al inicio, y así evitar la eliminación de variables fuera de rango.

Una vez hecho todo esto, se ha observado que aproximadamente un 50% de los errores que se producen en la respuesta de la RNA, son debidos a una excesiva longitud de las frases de salida, es decir, ponía más palabras de las esperadas en la solución del problema. Para solventar este problema, se ha creado una red previa a la anterior, que teniendo como entrenamiento las mismas entradas que la red principal, como salida solo posee un array numérico con la longitud de salida. Con las máximas longitudes de ficheros trabajadas, el tiempo necesario de entrenamiento de esta red no supera el medio minuto. Este tratamiento está implementado dentro de la función "DeleteExcedingWords".

También ha sido necesario crear una función propia para detectar si un número dado es par o impar. Es utilizada por AlphaNumericIn e InDimensions.

Una vez todas estas soluciones han sido implementadas, me he encontrado con que la red, una vez bien entrenada, tenía un error a la salida, hecho que me ha llevado a crear otra variable auxiliar, para que dijese exactamente donde se ha producido el error(o los errores). Esta variable se ha llamada VectorError y es creada por la función CheckErrors. Este registro de errores me ha permitido detectar un error en el texto de entrenamiento.

Para poder ejecutar el programa se ha de ejecutar simplemente la función main(), la cual está compuesta por las funciones del **código.1**(ver anexos).

2.8- Resultados finales

Este entrenamiento también ha sido ligeramente probado con silogismos, y muestra también una cierta capacidad para, dadas unas premisas, deducir una conclusión lógica, así que podríamos encontrarnos con un día en el que estuviesen lo suficientemente bien entrenadas como para solventarnos muchas tareas intelectuales.

Al trabajar con una matriz de 1000 frases y sus correspondientes 1000 frases de salida, los resultados son altamente esperanzadores. Con un tiempo inferior a media hora, trabajando con el equipo a 1,6Ghz se obtiene una RNA entrenada, capaz de traducir cualquiera de las mil frases, dadas a entrenamiento 900 de ellas escogidas aleatoriamente.

En el caso de trabajar con 2000 frases, es decir, 1900 de entrenamiento, la red muestra una mayor dificultad en el aprendizaje. Por eso, para ayudar a esta red, se ha realizado la nueva red supervisora de longitud “RNALongSelection”. Para ser un producto completo, debería alcanzar la capacidad de entrenarse correctamente con una cantidad posiblemente de varios millones de frases diferentes. La distancia por recorrer es todavía larga, hasta llegar a esa cantidad de entrenamiento satisfactorio, pero afortunadamente la forma en que se entrena es muy sencilla y permite la creación de conjuntos de entrada nuevos fácilmente.

Ante esta perspectiva, es posible que en el futuro, si se entrenase esta red con conjuntos de entrenamiento mayores, llegásemos a un punto donde estas redes no pudiesen solventar el problema. Entonces todavía quedaría una posibilidad: simplificar el trabajo de la red, en diferentes redes. Este problema, desde mi punto de vista puede ser abordado desde dos perspectivas, tratadas en el siguiente apartado de planificación (2.9).

El tiempo de entrenamiento de las 1900 frases para el Quad Core*2,6 GHz 2Gb de memoria RAM, ha sido de 500 segundos si se entrena de español a inglés. En cambio, al hacerlo de inglés a español, se ha podido observar la necesidad de añadir un 25% más de tiempo. La cantidad de neuronas se ha podido mantener estable a 250 neuronas en la capa oculta. Se ha podido observar que esta versión de Matlab no permite utilizar el 100% de la capacidad de las CPU’s utilizando tecnología “Quad Core”, y debido a este hecho, el tiempo solo debe duplicarse para el uso del Pentium IV, al no utilizar toolboxes especiales que permiten utilizar paralelismo.

2.9- Planificación

Al tener este proyecto las características de software y de investigación, no es posible realizar una línea de acción clara, con tiempo definidos, representables por un diagrama de Gantt o Pert. Por lo tanto, la planificación está basada en el desarrollo de las funciones que vayan solucionando los diferentes problemas a resolver, mostrados en los apartados 2.1.1 y 2.1.2, en el orden mostrados. Una vez funcionaban las funciones, se ha procedido a la mejora del programa, empezando por la versión 1.0 y hasta llegar a la versión 4.4, la versión más estable y completa, pasando por más de 20 versiones diferentes.

2.10- Orden de prioridad entre los documentos básicos

Aquí se expresan, en orden de prioridad, los documentos que deben ser tomados como válidos en caso de existir discrepancias entre ellos: código fuente, memoria, pliego de condiciones, anejos, planos, estudios con entidad propia, presupuesto, estado de mediciones.

2.11- Descripción detallada del programa

2.11.1- Descripción del programa main

El proceso del programa consiste en separar el archivo “In”, en entradas y en salidas. Una vez hecho esto, se realiza la conversión alfanumérica y una vez hecho esto, se guardan como números las palabras y se procede al entrenamiento. Una vez entrenada la red, se cuentan los errores y se reconvierte. Este programa debe satisfacer el almacenamiento de las variables requeridas en main2. Una vez obtenida la red entrenada deseada, se recomienda guardar el conjunto de variables obtenidas en “finalnet” (si se quiere traducir de la lengua principal a la secundaria) y en “finalnet2” en caso contrario, utilizando el comando: save *nombrearchivo.*

Por lo tanto, las variables introducidas, deben ser el archivo “in”, que contiene el conjunto de frases de entrenamiento. Estas deben ser alternadas: idioma1,idioma2,idioma1,…y cada una de ellas debe finalizar con un punto. Se reconoce que posee un sistema bastante sensible a los errores y es deseable una mejora de este aspecto, ya que si se introducen dos espacios seguidos se produce un error, ya que entiende que después de un espacio comienza una nueva palabra y coge dicho espacio como nueva palabra, creando la siguiente palabra con un espacio como predecesor.

La otra variable que debe ser cargada es *language*, con los valores “1” o “2”.
Para que las líneas impares del fichero de entrada ”in” resulten ser las de la entrada de la red, language se debe configurar a “1”, es decir:
-Si la primera frase introducida es en inglés y la segunda en español y “language=1”, la traducción deberá ser de inglés a castellano.
-Si las frases han sido introducidas de la misma manera, pero la solicitud de traducción la queremos de castellano a inglés, entonces se debe configurar “language=2”.

InDimensions:

La primera función en ejecutarse es “InDimensions”, que es la encargada de crear las siguientes variables:

- MaxLengthIn: Esta variable es la encargada de definir la máxima longitud de las frases de entrada (impares). Esto es necesario para la función “AlphaNumericConv”. Para hacer esto, se debe crear un doble bucle “for”, que examina cada letra y mira si está en una fila par y así diferencia entre lo que son salidas de la red y lo que son entradas.
- MaxLengthOut: Exactamente lo mismo que la anterior, pero con las frases de salida (pares).
- words: Aquí se almacenan todas las nuevas palabras que detecta, del idioma principal (idioma de entrada).
- words2: Lo mismo que la anterior variable, pero para el idioma secundario (pares).

Esta función también se encarga de invertir el orden de las frases si “language=2”, mediante la función “InvertOrder”.

AlphaNumericConv:

Se encarga de realizar la conversión alfanumérica, esto es realizado mediante un bucle "for" y aquí es donde se hace realmente la separación, en "AlphaNumericIn" y "AlphaNumericOut", por lo tanto también invoca a la función "PairDetection".

WordSeparation:

Esta tal vez sea la función más complicada, así que procederé a su explicación con cautela. El primer hecho notable en esta función es que repite todo el mismo proceso que realiza con "AlphaNumericIn", con "AlphaNumericOut". Así que explico solo el caso de la entrada, que es el primer caso.

Con el uso de los dos bucles "for" para recorrer toda la matriz de "AlphaNumericIn" carácter a carácter, se analiza primero de todo que sea diferente de cero, es decir, que no haya acabado. Y luego se almacenan los caracteres por los cuales se va pasando en un buffer de memoria llamado "palabraTemp". Se procede a llenar el buffer mientras no haya acabado la palabra actual, lo cual es considerado analizando el carácter. Si el carácter es un espacio, una coma o un punto, se considera que se ha finalizado la palabra y en ese momento llamamos a la función "CheckWord". Si no es ninguno de esos casos, continuamos llenando el búffer.

Esta función también se encarga de acabar de adecuar a nuestras necesidades las variables "finalsentence" y "finalsentenceOut", que son las variables que devuelve esta función.

CheckWord:

Aquí se analiza "palabraTemp", y se comprueba si existe en el archivo "words"; si no está creada, se crea en la última fila disponible, pero en ambos casos se obtiene el índice de la palabra temporal ("tempindexword") y se inserta en "finalsentence". Para hacer este proceso se requiere de un "flag", para evitar la creación de una palabra ya existente.
La reinicialización de las variables en el lugar preciso ha resultado un elemento crítico en esta función, conjunta y compenetradamente con "WordSeparation".

CrossValidation:

En este programa tenemos ya convertidos los ficheros de entrada y de salida, pero para comprobar el buen funcionamiento de la red, no debemos entrenarla con todos los datos disponibles, sino que optamos por elegir aleatoriamente 100 frases y estas no ser entrenadas. Las variables que se encargarán de tener todo el conjunto menos estas 100 frases son llamadas: "finalsentenceReduced" y "finalsentenceOutReduced".

Procedo solo a explicar el caso de "finalsentenceReduced": realizamos un volcado de "finalsentence" a "finalsentenceReduced". Creamos un "array" de números aleatorios y no repetitivos. Procedemos a su ordenación mediante la función específica de Matlab "sortrows". Una vez realizado esto, solo nos falta eliminar las líneas de frases que corresponden con los números aleatorios generados.

GetCounterOut:

Esta función tiene el propósito de hallar la variable "CounterOut", que consiste en un array que almacena la longitud de todas las frases de salida del entrenamiento. Esta variable servirá para entrenar la RNA secundaria, la encargada de averiguar la longitud de la frase.

RNALongSelection:

La misión de esta RNA, es de tener la misma entrada que la RNA principal, pero en lugar de tener la traducción de la entrada como salida, solo posee su longitud.
Esto se ha visto especialmente desde que se ha entrenado con casos en los cuales la longitud de la frase de entrada es diferente de la de salida, veamos un ejemplo:

yo conduzco unas bicicletas lentas. *I drive slow bikes.*

La frase en castellano ocupa cinco palabras, mientras en inglés solo cuatro.
Para hallar todo el array de longitudes, utilizamos la función "GetCounterOut". Una vez tenemos esta variable solo nos hace falta definir los parámetros de la red, que al ser sencilla, con un tiempo de 60 segundos, iremos sobrados y el número de neuronas en la capa oculta, para el caso de las 2000 frases, será de 50. Utilizaremos la variable "finalsentenceReduced" como entrada y "CounterOut" como salida, y la manera de introducir este código es la siguiente:

```
net=train(net,finalsentenceReduced,CounterOut);
```

Una vez entrenada, se procede a su simulación, mediante "finalsentence", para así obtener "FinalAutoLengthAdjustment", que nos será útil a posteriori, en la función "DeleteExcedingWords":

```
a = sim(net,finalsentence);
FinalAutoLengthAdjustment=int16(a);
```

Esta última función es para pasar a enteros los números racionales que ha calculado la red, ya que para reconvertirlas en palabras deben ser números enteros.

RNA:

Aquí es donde pasamos finalmente al entrenamiento y simulación de la red, sin mucho que destacar, excepto que se ha procedido a realizar una limpieza de todas las variable del "workspace" de Matlab para su posterior carga, excepto las indispensable para realizar el entrenamiento, ya que este proceso es un elemento crítico a la hora de consumir requisitos al sistema (especialmente memoria RAM).
El proceso es el siguiente: guardamos todas las variables en "PostLoad" (memoria ROM), eliminamos todas las variables, entrenamos la red y volvemos a cargar "PostLoad".

La RNA es entrenada mediante las variables “finalsentenceReduced” y “finalsentenceOutReduced” de la siguiente manera:

net=train(net,finalsentenceReduced,finalsentenceOutReduced);

Y simulada así:

a = sim(net,finalsentence);

Por lo tanto entrenamos con una matriz de 5*1900 y simulamos con una de 5*2000 para comprobar errores.
El resultado es convertido a enteros y colocado correctamente mediante la transpuesta en “RNAAnswer”.

DeleteExcedingWords:

En esta función se trata de recortar “RNAAnswer”, es decir, modificarlo borrando las palabras que excedan de la longitud de frase calculada con la red secundaria, es decir, utilizando el array “FinalAutoLengthAdjustment”. Utilizamos dos bucles “for” para poner a “1” todas las casillas sobrantes, ya que el número “1” es la primera casilla, reservada en la memoria de “words”, para el espacio.

Aquí también se comprueba que no haya ninguna palabra fuera de rango. Este caso se daría si por un error, el resultado de una palabra de la red hiciera referencia a un número mayor al índice mayor de “words”. Por ejemplo:

Si existen 40 palabras guardadas en “words”, pero la respuesta de la red en una casilla es de 50, esa palabra no podrá ser reconvertida.

Para solucionar este problema se hace que si la palabra (RNAAnswer(n,m)), es mayor que el número máximo de palabras (maxwords), entonces RNAAnswer(n,m)=maxwords.
Si RNAAnswer(n,m) es menor que 1, entonces se pone a “1”.

ConvWords:

Dado que tanto el archivo “words” como “words2” están codificados de tal manera que cada palabra es representada por sus caracteres ASCII, su traducción por parte del usuario es costosa. Para poder comprobar la correcta creación de estos archivos, utilizamos esta función para tener en formato de palabras con lenguaje natural en los archivos: “wordsconverted” y “wordsconverted2”.

CheckErrors:

Una vez tenemos las variables “RNAAnswer” y “finalsentenceOut”, es sencillo compararlas para contabilizar los errores. Para ello ha sido muy útil la siguiente función, predefinida por el Matlab: isequal. Empleada de la siguiente manera:

```
equal = isequal(finalsentenceOut(n,:),RNAAnswer(n,:));
```

Reconversion:

Y ahora ya solo queda reconvertir las frases que ha devuelto la red, en la variable "RNAAnswer", lo cual se realiza sin dificultades, mediante los índices de "words" para obtener las palabras y luego realizando la conversión a tipo de variable "char", es decir, carácter.

2.11.2- Descripción del programa main2

Main2

Al ser simple la misión restante, se ha optado por realizarlo todo dentro del *main*, llamando solamente a tres funciones, así resulta muy sencillo para el usuario cambiar cualquier parte. Todavía no se ha instalado ninguna función para designar idioma, por lo tanto, para utilizar este programa, el usuario deberá seleccionar el idioma. Esto se realiza mediante la variable "language", la misma que en el primer programa, solo que aquí tiene una ligera diferente connotación.

Si "language=1", la pregunta deberá ser formulada con el lenguaje principal, en este caso el inglés.
Por lo tanto, el lenguaje secundario corresponde a "language=2".

Hay que tener en cuenta que en función del idioma, se necesita una red neuronal entrenada en un sentido o el contrario, por ese motivo, en función del idioma se carga automáticamente la red apropiada entrenada con las 1900 frases. Por lo tanto, para introducir una nueva red, se debe hacer después de que el programa haya cargado la suya o suprimiendo las dos líneas de código que hacen referencia a esta carga, estas son: "*load finalnet"* y "*load fínanlet2"*.

El programa también carga automáticamente una pregunta, por lo que el usuario, para hacer su propia pregunta debe hacerlo en la variable "ask".

Como en el "main" primero, aquí deberemos hallar "finalsence", que será la conversión de "ask". Para ello utilizamos la función "WordSeparationAsk", que es la encargada de establecer los índices de palabra que obtiene de "words", es muy parecida a la anteriormente utilizada "WordSeparation", aunque más simple debido a que ahora se trata de un array, en lugar de una matriz, de elementos a convertir. Aunque la longitud de esta reconversión habrá de ser igual al número de neuronas en la capa de entrada y para este propósito, se crea primero una matriz con todo unos, que será la encargada de almacenar la frase con su correcta longitud, en el archivo "in".
A continuación ya se puede simular con la siguiente instrucción:
RNAAnswer = sim(net,in);
Una vez tenemos la respuesta de la red, podemos adecuar su longitud, como se había hecho anteriormente utilizando la función "DeleteExcedingWords()".

Y para finalizar solo queda reconvertir la respuesta a lenguaje natural mediante la función "Reconversión", explicada anteriormente.

2.12- Perspectivas de futuro

El hecho de trabajar en la traducción automática mediante RNAs, que es un problema abordable pero de alta complejidad, implica la utiliazación de largos ciclos de cálculo. y que, por lo tanto, requiere largos tiempos de cálculo. El tiempo que tardemos los humanos en utilizarlas dependerá de la potencia informática que dispongamos y el interés que le dediquemos a ello, para invertir unos ciertos recursos en ello y así poder realizar los códigos adecuados a nuestras necesidades.

A pesar de no haber logrado crear un algoritmo definitivo, ni mucho menos, se ha podido establecer una base sobre la cual poder trabajar, generalizar, mejorar y crear un entorno simple de usuario.

Se ha comprobado que el entrenamiento *palabra a palabra* resulta muy práctico a la hora de introducir conocimientos básicos que a la larga podrían ir recopilándose para recoger el conocimiento humano.

Para solventar el obstáculo de proponer a una red neuronal un problema demasiado grande, podemos intentar resolverlo de esta manera:

- Crear redes entrenadas con dos mil frases. Cada vez que vengan nuevas frases se deberán colocar en redes neuronales abiertas (con menos de mil frases), según el número de palabras coincidentes en el archivo “words”. Esta opción tiene un serio inconveniente, se necesitará un tamaño del conjunto de entrenamiento muy grande, para lograr que de la frase preguntada, se encuentren íntegramente todas las palabras.

Así pues, se podría entrenar un conjunto de redes con conocimientos de medicina para enseñarle, mediante lenguaje natural, los síntomas de un paciente y que sugiera hipótesis sobre la/s enfermedad/es o utilizarla de consulta de enfermedades, de medicamentos o cualquier otra cuestión que haya sido entrenada para tal. Tendríamos pues una evolución hacia un sistema experto. También se podría mejorar el proyecto, con la implantación de filtros voz-caracteres y caracteres-voz para mantener diálogos, en vez de tener que escribir.

En cualquier ámbito, ya sea de ingeniería, historia, geología, o cualquier otro campo se podría aplicar como base de datos interactiva, así como montar un sistema para el fácil entrenamiento de las redes, el cual requeriría de una cierta atención por parte del usuario, con un menú configurado.

En definitiva, entre tanta programación no logro encontrar la tarea para la cual esta herramienta, una vez desarrollada satisfactoriamente, no tuviese aplicación.

Unos cálculos aproximados apuntan que en el período de un año, con el único apoyo de un ingeniero informático (que podría ser de último año de carrera), se podría tener realizado un estudio completo de viabilidad, y en caso de ser dicho estudio favorable, en el período de otro año, tener listo un programa funcional para el público, con diferentes bases de datos, para poderse descargar en un bloque entero o fragmentos temáticos parciales. Pudiéndose sacar un beneficio económico posterior mediante la publicidad de la página

web de descarga. Con un equipo del doble, es decir, de tres informáticos, los tiempos se estiman a la mitad y siguiendo yo solo el tiempo sería de dos años.

Así que animo a cualquier persona con capacidad de conseguir proyectos con fondos económicos, a presentar dicha petición, para dirigir este proyecto, que a mínimas expectativas produciría interesantes resultados sobre los que realizar futuras investigaciones.

Traducción automatizada basada en algoritmos de redes neuronales artificiales

PLANOS

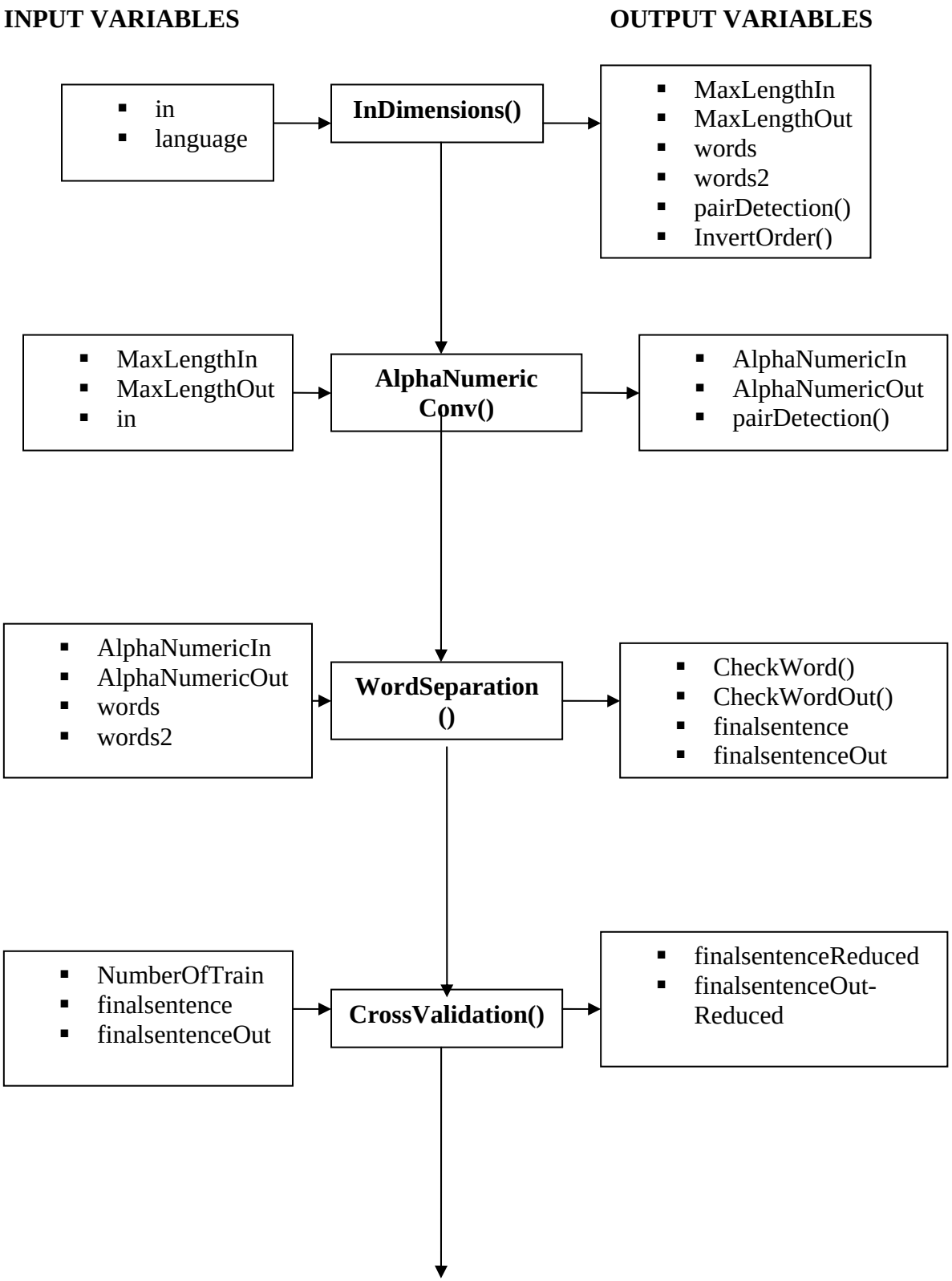
Titulación: Enginyeria Tècnica Industrial en Electrònica Industrial

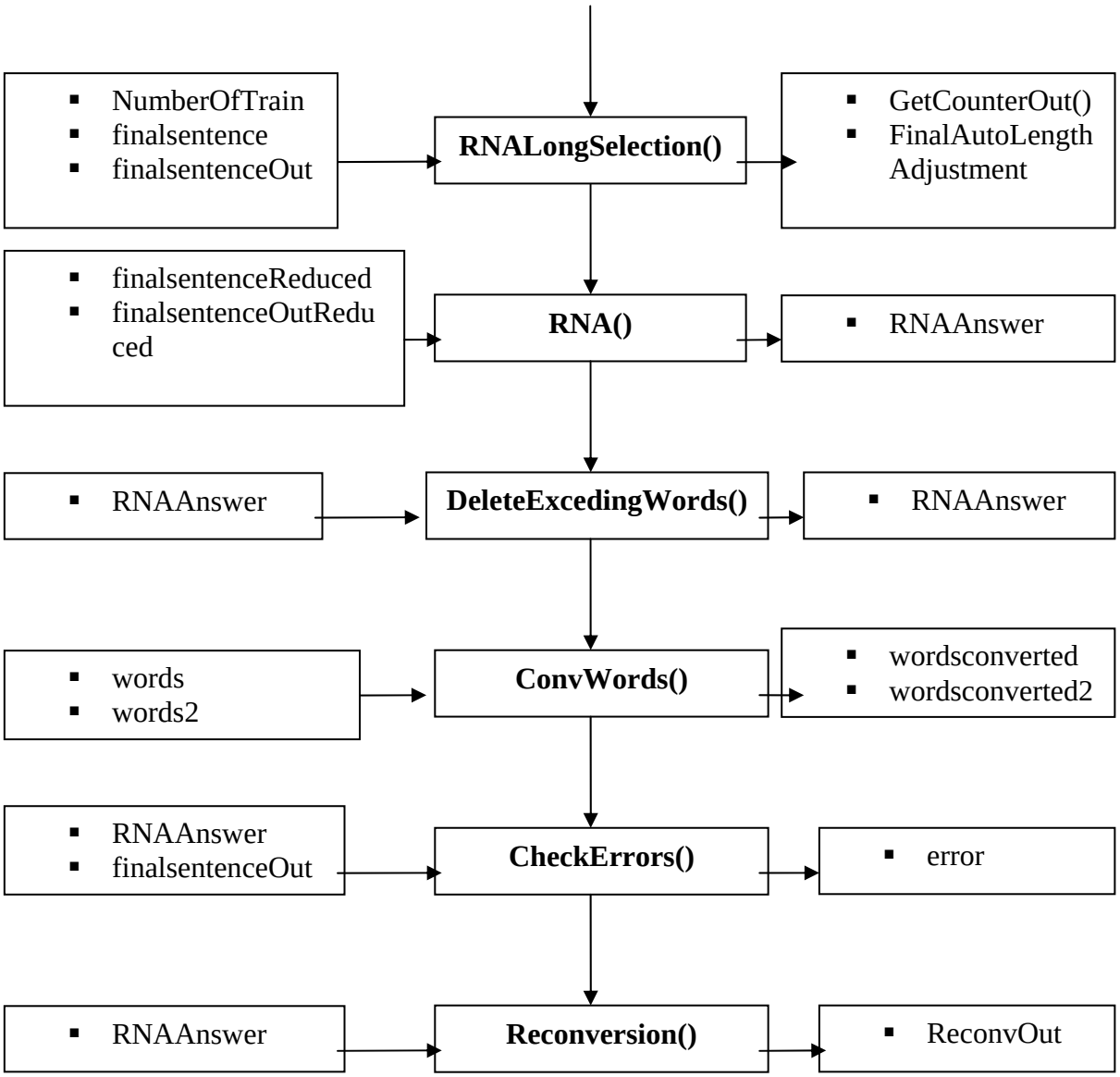
AUTOR: Xavier Escobar
DIRECTORES: Jesús Brezmes, Eduard Llobet

Fecha: Junio / 2009 .

3.- Esquemas

Este diseño de esquema, cuyo diseño ha sido inventado para este propósito, pretende dar una rápida comprensión general del código en cuestión. En el tronco del esquema, se visualizan las funciones que va llamando main(). A su izquierda, las variables que posee como elementos de entrada a la función; a su derecha las variables de retorno.




NumberOfTrain
finalsentence
finalsentenceOut
RNALongSelection()
GetCounterOut()
FinalAutoLength Adjustment
finalsentenceReduced
finalsentenceOutReduced
RNA()
RNAAnswer
RNAAnswer
DeleteExcedingWords()
RNAAnswer
words
words2
ConvWords()
wordsconverted
wordsconverted2
RNAAnswer
finalsentenceOut
CheckErrors()
error
RNAAnswer
Reconversion()
ReconvOut

Traducción automatizada basada en algoritmos de redes neuronales artificiales

ESTADO DE MEDICIONES

Titulación: Enginyeria Tècnica Industrial en Electrònica Industrial

AUTOR: Xavier Escobar
DIRECTORES: Jesús Brezmes, Eduard Llobet

Fecha: Junio / 2009 .

4.- Estado de mediciones

Aquí se ha procedido a listar las diferentes unidades de obra presupuestables, organizadas en tres capítulos:

AMIDAMIENTOS

Codigo	Ud	Descripción	Cantidad
		CAPÍTULO C_01 PREPARACIÓN	
1	u	Instalación sistema operativo	3,00
2	u	Instalación programa Matlab	3,00
		CAPÍTULO C_02 CONSTRUCCIÓN	
3	u	Creación de la matriz de datos de entrada	1,00
4	u	Creación de los programas	1,00
		CAPÍTULO C_03 ACABADO	
5	u	Redacción de documentación	1,00
6	u	Desarrollo y mejoras	1,00

Traducción automatizada basada en algoritmos de redes neuronales artificiales

PRESUPUESTO

Titulación: Enginyeria Tècnica Industrial en Electrònica Industrial

AUTOR: Xavier Escobar
DIRECTORES: Jesús Brezmes, Eduard Llobet

Fecha: Junio / 2009 .

5.- Presupuesto

Índice del presupuesto:

5.1 Precios unitarios

Aquí se listan los precios unitarios, que servirán más adelante para confeccionar las unidades de obra; las unidades son mostradas en horas o unidades individuales.

LISTA DE PRECIOS UNITARIOS				
Código	Ud	Descripción	Precio	
7	u	Programa Matlab	120,00	CIENTO VEINTE EUROS con CERO CÉNTIMOS
8	h	Becario universidad	6,50	SEIS EUROS con CINCUENTA CÉNTIMOS
9	h	Amortización de Hardware	3,00	TRES EUROS con CERO CÉNTIMOS

5.2 Cuadro de descompuestos

Se procede a continuación al desglose de las diferentes unidades de obra, teniendo en cuenta la mano de obra, la maquinaria y los materiales necesarios para conformar cada unidad.

CUADRO DE DESCOMPUESTOS

Código	Cantidad Ud	Descripción	Precio	Subtotal	Importe
CAPÍTULO C_01 PREPARACIÓN					
1	**u**	**Instalación sistema operativo**			
8	**1 h**	Becario universidad	6,50	6,50	
9	**1 h**	Amortización de Hardware	3,00	3,00	
		Suma la partida.....			9,50
		Costos indirectos.....2,00%			0,19
			TOTAL PARTIDA.....		9,69
Sube el precio total de la partida a la mencionada cantidad de VEINTI NUEVE EUROS con SIETE CÉNTIMOS.					
2	**u**	**Instalación programa Matlab**			
8	**1 h**	Becario universidad	6,50	19,50	
9	**1 h**	Amortización de Hardware	3,00	9,00	
7	**1 u**	Programa Matlab	120,00	360,00	
		Suma la partida.....			388,50
		Costos indirectos.....2,00%			7,77
		TOTAL PARTIDA.....			396,27
Sube el precio total de la partida a la mencionada cantidad de TRES CIENTOS NOVEINTA Y SEIS EUROS con VEINTISIETE CÉNTIMOS.					
TOTAL CAPÍTULO C_01 PREPACIÓN				……..	405,96

CAPÍTULO C_02 CONSTRUCCIÓN					
Código	Cantidad Ud	Descripción	Precio	Subtotal	Importe
3	**u**	**Creación de la matriz de datos de entrada**			
8	**30 h**	Becario universidad	6,50	195,00	
9	**30 h**	Amortización de Hardware	3,00	90,00	
			Suma la partida.....		285,00
			Costos indirectos.....2,00%		5,70
			TOTAL PARTIDA.....		290,70
Sube el precio total de la partida a la mencionada cantidad de DOS CIENTOS NOVEINTA EUROS con SETENTA CÉNTIMOS.					
4	**u**	**Creación de los programas**			
8	**800 h**	Becario universidad	6,50	5.200,00	
9	**800 h**	Amortización de Hardware	3,00	9,00	
			Suma la partida.....		5.209,00
			Costos indirectos.....2,00%		104,18
			TOTAL PARTIDA.....		5.313,18
Sube el precio total de la partida a la mencionada cantidad de CINCO MIL TRESCIENTOS TRECE EUROS con DIECIOCHO CÉNTIMOS.					
TOTAL CAPÍTULO C_02 CONSTRUCCIÓN				……..	5.603,88

CAPÍTULO C_03 ACABADO					
3	**u**	**Redacción de documentación**			
8	**70 h**	Becario universidad	6,50	195,00	
9	**70 h**	Amortización de Hardware	3,00	90,00	
			Suma la partida.....		285,00
			Costos indirectos.....2,00%		5,70
			TOTAL PARTIDA.....		290,70
Sube el precio total de la partida a la mencionada cantidad de DOS CIENTOS NOVEINTA EUROS con SETENTA CÉNTIMOS.					
4	**u**	**Desarrollo y mejoras**			
8	**100 h**	Becario universidad	6,50	650,00	
9	**100 h**	Amortización de Hardware	3,00	9,00	
			Suma la partida.....		659,00
			Costos indirectos.....2,00%		13,18
			TOTAL PARTIDA.....		672,18
Sube el precio total de la partida a la mencionada cantidad de SEISCIENTOS SETENTA Y DOS EUROS con DIECIOCHO CÉNTIMOS.					
TOTAL CAPÍTULO C_03 ACABADO				…….	962,88

5.3 Presupuesto

Como consecuencia del cuadro de descompuesto, aquí aparece el presupuesto general.

PRESUPUESTO				
Código	Descripción	Cantidad	Precio	Importe
CAPÍTULO C_01 PREPARACIÓN				
1	Instalación sistema operativo	3	9,69	29,07
2	Instalación programa Matlab	3	405,96	1.217,88
TOTAL CAPÍTULO C_01 INSTALACIÓN			……..	1.246,95
CAPÍTULO C_02 CONSTRUCCIÓN				
3	Creación de la matriz de datos de entrada	1	290,70	290,70
4	Creación de los programas	1	5.313,18	5.313,18
TOTAL CAPÍTULO C_02 CONSTRUCCIÓN			……	5.603,88
	CAPÍTULO C_03 ACABADO			
5	Redacción de documentación	1	290,70	290,70
6	Desarrollo y mejoras	1	3.763,80	5.313,18
TOTAL CAPÍTULL C_03 ACABADO			……	5.603,88
TOTAL		……		12.454,71

5.4 Resumen del presupuesto

A modo de síntesis, quedan simplificados aquí los resultados del presupuesto:

RESUMEN DE PRESUPUESTO

Capítulo	Resumen	Importe	%
C_01	PREPARACIÓN	1.246,95	15,96
C_02	CONSTRUCCIÓN	5.603,88	71,72
C_03	ACABADO	962,88	12,32
	TOTAL EJECUCIÓN	7.813,71	
	13,00 % Gastos Generales……882,66		
	6,00 % Beneficio industrial……468,82		
	SUMA DE G.G. y B.I.	1.351,48	
	16,00 % I.V.A……..9939,89	1.466,43	
	TOTAL PRESUPUESTO GENERAL	10.631,62	

Sube el presupuesto general a la mencionada cantidad de DIEZ MIL SEISCIENTOS TREINTA Y UN EUROS con SESENTA Y DOS CÉNTIMOS

Traducción automatizada basada en algoritmos de redes neuronales artificiales

PLIEGO DE CONDICIONES

Titulación: Enginyeria Tècnica Industrial en Electrònica Industrial

AUTOR: Xavier Escobar
DIRECTORES: Jesús Brezmes, Eduard Llobet

Fecha: Junio / 2009 .

6.- Pliego de Condiciones

Índice del pliego de condiciones:

6.1.- Comentario.

Dado el carácter informático del proyecto, no existen condiciones técnicas o ambientales que puedan afectarle o referirse al mismo de forma alguna. No obstante las condiciones generales vienen impuestas del exterior, como consecuencia de las exigencias que se piden en su realización.

6.1.1.- Reseña.

Las exigencias que limitan la construcción del proyecto de aplicación informática, son las siguientes:

-Utilización del programa de simulación MATLAB.

El proyecto a realizar debería resultar lo más universal posible, para su utilización en cualquier medio informático muy expandido en el mercado; no precisando modificación alguna en su instalación para ordenadores compatibles.

6.2.- Condiciones recomendadas de utilización.

En la utilización del proyecto de aplicación informática se derivan una serie de condiciones que limitan de alguna forma la utilización del mismo. Estas condiciones resultan como consecuencia de llevar a la práctica la realización del proyecto.
Son las siguientes:

6.2.1.- Condiciones software:

Otro requerimiento obvio muy necesario es estar en posesión del programa simulador MATLAB, en su versión comercial en una versión 7.00 o posterior.

Para dicho programa hace falta tener instalado el sistema operativo, ya sea Windows, Solaris, Linux o Macintosh.

6.2.2.- Condiciones hardware:

Estos requerimientos se refieren al soporte físico mínimo para el correcto funcionamiento
Al tratarse de un sistema multiplataforma, los requisitos dependerán del sistema operativo.

Windows

Operating Systems	Processors	Disk Space	RAM
32-bit MathWorks Products			
Windows XP Service Pack 2 or 3 Windows Server 2003 Service Pack 2 or R2 Windows Vista Service Pack 1 Windows Server 2008	Intel Pentium 4 and above Intel Celeron* Intel Xeon Intel Core AMD Athlon 64* AMD Opteron AMD Sempron*	680 MB** (MATLAB only)	512 MB (At least 1024 MB recommended)
64-bit MathWorks Products			
Windows XP x64 Service Pack 2 Windows Server 2003 x64 Service Pack 2 or R2 Windows Vista Service Pack 1 Windows Server 2008	Intel Pentium 4 and above Intel Celeron* Intel Xeon Intel Core AMD64	680 MB** (MATLAB only)	1024 MB (At least 2048 MB recommended)

Linux

Operating Systems	Processors	Disk Space	RAM
32-bit MathWorks Products			
Debian 4.0 and above Red Hat Enterprise Linux v.4 and above OpenSuSE 9.3 and above Ubuntu 8 and above Other distributions*	Intel Pentium 4 and above Intel Celeron** Intel Xeon Intel Core AMD Athlon 64** AMD Opteron AMD Sempron**	500 MB (MATLAB only)	512 MB (At least 1024 MB recommended)
64-bit MathWorks Products			
Debian 4.0 and above Red Hat Enterprise Linux v.4 and above OpenSuSE 9.3 and above Ubuntu 8 and above Other distributions*	Intel Pentium 4 and above Intel Celeron** Intel Xeon Intel Core AMD64	500 MB (MATLAB only)	1024 MB (At least 2048 MB recommended)

* For any 32-bit or 64-bit Linux distribution not explicitly listed above, R2009a requires the distribution to be built using Kernel 2.4.x or 2.6.x and glibc (glibc6) 2.3.4 and above.

Solaris

Operating Systems	Processors	Disk Space	RAM
64-bit MathWorks Products			
Solaris 10*	ultraSPARC	700 MB (MATLAB only)	1024 MB (At least 2048 MB recommended)

Macintosh

Operating Systems	Processors	Disk Space	RAM
32-bit MathWorks Products			
Mac OS X 10.5.5 and above	All Intel-based Macs	360 MB (MATLAB only)	512 MB (At least 1024 MB recommended)

6.2.3.- CONDICIONES GENERALES:

Son las siguientes, y afectan a la utilización correcta del usuario con el conjunto de aplicación:
-Tanto los archivos de datos como las funciones, deben ser contenidas en su totalidad en la misma carpeta y dicha carpeta debe ser situada en el “current directory” de Matlab.

-Para variar el fichero de entrada, debe ejecutarse la orden: “load preload”. Se cargarán tres variables: in, textcomplete y language. El contenido de in, debe ser modificado muy minuciosamente y siguiendo el mismo formato. Dicho contenido debe ser copiado a textcomplete por el usuario.

-El contenido de la variable language debe ser “1” o “2” y es utilizado para alternar las frases de entrada por las de salida.

El correcto seguimiento de estas condiciones, asegura que los resultados obtenidos por el programa alcanzan un nivel de fiabilidad alto.

Traducción automatizada basada en algoritmos de redes neuronales artificiales

ANEJOS

Titulación: Enginyeria Tècnica Industrial en Electrònica Industrial

AUTOR: Xavier Escobar
DIRECTORES: Jesús Brezmes, Eduard Llobet

Fecha: Junio / 2009 .

7.- Anejos

7.1. Documentación de partida: Redes Neuronales

A continuación procedo a realizar una breve explicación sobre las RNA, explicando especialmente los aspectos que afectan a este proyecto. Estas, deben ser entendidas como un conjunto de algoritmos informáticos que pretenden variar parámetros de una función llamada "net", en función de las entradas y las salidas aplicadas, para obtener una función de transferencia que satisfaga nuestras necesidades, proporcionando predicciones a entradas nuevas. Al ser entrenada, la red será capaz de dar una respuesta "lógica" a una entrada desconocida. Se suele decir que las redes neuronales son aproximadores universales a relaciones multivariantes entre funciones/valores de entrada y de salida.

Una definición altamente aceptada entre los entendidos del tema, ha sido dada por Robert Hetch-Nielsen y entiende la red neuronal del siguiente modo:

Sistema de computación que consta de un gran número de elementos simples, muy interconectados, que procesan la información respondiendo frente a unos estímulos externos.

La red está organizada por capas, existiendo tres tipos:

1.- Capa de entrada: su número de neuronas viene definido por los datos de entrada del problema.

2.- Capa oculta: consiste en una capa intermedia, que pueden ser varias capas ocultas, pero que a la práctica, no se han demostrado evidencias de que esto suponga una mejora para el aprendizaje.

3.- Capa de salida: su número depende de los datos de salida esperados.

En este proyecto nos hemos centrado en la utilización de la red perceptrón multicapa. En las RNA de tipo perceptrón multicapa, la unidad básica se denomina neurona y recibe unas entradas (valores numéricos) a las que realiza una operación (generalmente suma) y al resultado se le aplica una función, tambien llamada función de activación. Esta función devuelve otro resultado que envía a más neuronas, hasta llegar a la salida final. La idea general está inspirada en el sistema nervioso animal, siendo una pobre imitación del mismo, pero muy útil para nuestros propósitos.

El proceso de adaptación de la función se realiza básicamente mediante los pesos, que son factores numéricos, que servirán para ir multiplicando las entradas y así ir obteniendo los valores propagados. Normalmente estos pesos se denominan con la letra W (viene de la palabra inglesa weights). Aquí tenemos un sencillo esquema de cómo se distribuyen la entrada, la salida, los pesos y la capa oculta, que sirve simplemente para tener más pesos para entrenar y modificar la función de transferencia:

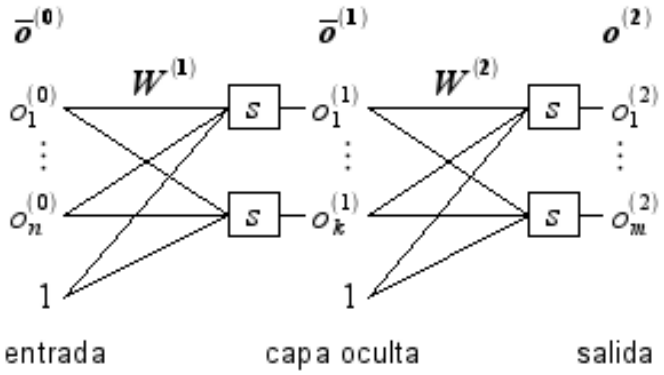


Figura 7.1.

Los elementos que constituyen a la red neuronal son los siguientes:

- *s* es una función de valores reales, conocida como la función de transferencia.
- $\bar{o}^{(0)}$ es la capa de entrada, considerado como el vector extendido del ejemplo $o^{(0)} = x = (x_1, \ldots, x_n)^T$.
- $\bar{o}^{(1)}$ es la capa oculta, el vector extendido de $o^{(1)} = (o_1^{(1)}, \ldots, o_k^{(1)})^T$.
- $o^{(2)} = (o_1, \ldots, o_m)^T$ es la capa de salida, considerado como el vector que aproxima al valor deseado $t = (t_1, \ldots, t_m)^T$.
- $W^{(1)}$ es una matriz de tamaño $(n+1) \times k$ cuyos valores $W_{ij}^{(1)}$ son los pesos de la conexión entre las unidades $\bar{o}_i^{(0)}$ y $o_j^{(1)}$.
- $W^{(2)}$ es una matriz de tamaño $(k+1) \times m$ cuyos valores $W_{ij}^{(2)}$ son los pesos de la conexión entre las unidades $\bar{o}_i^{(1)}$ y $o_j^{(2)}$

La función de activación consiste en convertir la entrada neta de la net "net_k", en un nivel de activación para la neurona, realizando una equivalencia al nivel de excitación de una neurona biológica. La función de activación, compara la entrada total, con un umbral propio y controla la activación de la neurona.
Ya que una RNA es un sistema dinámico que evoluciona en el tiempo, para su mayor efectividad, debemos ser capaces de expresar este sistema en ecuaciones diferenciales, por lo tanto, la función de activación ha de ser derivable. Por ese motivo no se utiliza la función escalón, debido a su discontinuidad y se utiliza normalmente la sigmoide, mostrada a continuación:

$$f(net_k) = \frac{1}{1 + e^{-\lambda \cdot net_k}}$$

Figura 7.2.

Figura 7.3.

Existen gran variedad de tipos de redes neuronales, algunas muy rápidas, pero poco inteligentes, otras muy inteligentes pero lentas. Así que en lugar de estar probando con cada uno de los tipos, nos centramos en el que es considerado el aproximador universal: la red perceptrón multicapa entrenada con el algoritmo de retropropagación del gradiente del error (backpropagation multilayer perceptron).

A grandes rasgos el algoritmo realiza los siguientes pasos:

1. Calcular la salida de la red $\boldsymbol{o}^{(2)}$ a partir de uno de los conjuntos de valores de prueba *x*.
2. Comparar con la salida correcta *t* y calcular el error según la fórmula:

$$E(\boldsymbol{x}; \boldsymbol{W}^{(1)}, \boldsymbol{W}^{(2)}) = \frac{1}{2} \sum_{i=1}^{m} (o_i^{(2)} - t_i)^2.$$

3. Calcular las derivadas parciales del error con respecto a los pesos $\boldsymbol{W}^{(2)}$ que unen la capa oculta con la de salida.
4. Calcular las derivadas parciales del error con respecto a los pesos $\boldsymbol{W}^{(1)}$ que unen la capa de entrada con la oculta.
5. Ajustar los pesos de cada neurona para reducir el error.
6. Repetir el proceso varias veces por cada par de entradas-salidas de prueba.

A pesar de la gran cantidad de elementos (variables) que conforman una red y que pueden ser configurados por el usuario, en la práctica, para obtener una buena red capaz de resolver nuestro problema en un tiempo razonable, hemos optado por dejar todos los parámetros constantes excepto el tiempo de entrenamiento y el número de neuronas en la capa oculta. Si el número de neuronas es mayor, también se incrementa la capacidad de elementos que puede aprender, pero está demostrado que un exceso de ellas, puede provocar efectos adversos; por este motivo ese número debe ser buscado minuciosamente y con paciencia ya que no existe ninguna función ni método directo para calcular sus dimensiones óptimas.

Esta tecnología permite desarrollar una gran capacidad computacional, debido a su estructura en paralelo y, por lo tanto, un tratamiento simultáneo de los datos. Este sistema de RNA’s, tiene también una gran tolerancia a fallos y no requiere de una descripción total del problema, así como no tiene una ruptura total del programa, sino una función de error continua.

7.2 Anejos de aplicación en el ámbito del proyecto

7.2.1 Códigos main()

```
% Main (General Trainning)

%First program to RNA analisis and trainning
InDimensions()%Create words,MaxLengthIn,MaxLengthOut
AlphaNumericConv()%Create AlphaNumericIn and AlphaNumericOut
WordSeparation()%It returns "finalsentence" and "finalsentenceOut"
CrossValidation()%It returns "finalsentenceReduced" and
"finalsentenceOutReduced"
RNALongSelection()%It returns FinalAutoLengthAdjustment
RNA()%Returns RNAAnswer
DeleteExcedingWords()%Returns RNAAnswer (upgraded)
ConvWords()%Just to see the words and words2 in human format
%It returns: wordsconverted and wordsconverted2
CheckErrors()%Error
Reconversion()%It returns ResAsk
error%Show the number of errors
```

Código 1. Main.

```
%Script AlphaNumericConv
%This script creates the variable "AlphaNumericOut", this variable is the
%numeric ASCII conversion of the variable "in".

AlphaNumericOut =[];
AlphaNumericIn =[];
textTemp=[];%This variable will save the content of one of the rows of
the
            %input text.

for n=1:length(in);
    textTemp = cell2mat(in(n));%textTemp gets the string characters of a
row
    textTemp = double(textTemp);%One number is gived for each different
character
    InputPair=n;
    PairDetection()%Script to know if a number is pair or odd.
%This is for separate in different variables the input and output.
    if pair==0
        AlphaNumericIn=[AlphaNumericIn;textTemp,zeros(1,MaxLengthIn-
length(textTemp))];
    else
        AlphaNumericOut=[AlphaNumericOut;textTemp,zeros(1,MaxLengthOut-
length(textTemp))];
    end
 end
%This variable gets his previous stat plus the converted array row of
numbers.

%clear the not neededs buffers of memory
clear textTemp
clear n
clear pair
clear InputPair
clear MaxLengthIn
clear MaxLengthOut
%Process='The AlphaNumericConv Function is passed correctly'
```

Código 2. AlphaNumericConv.

```
%Script CheckErrors

%ResAsk=[];
%TextSelected=[];
finalsentenceOut=finalsentenceOut';
error=0;
VectorError=[];
equal=0;
[RNAAnswerY,RNAAnswerX]=size(RNAAnswer);
[finalsentenceOutY,finalsentenceOutX]=size(finalsentenceOut);

%Comparison of matrix
for n=1:finalsentenceOutY;
        equal = isequal(finalsentenceOut(n,:),RNAAnswer(n,:));
        if equal==0
            error=error+1;
            VectorError(1,error)=n;
        end
end

clear finalsentenceOut
clear equal
clear n
clear finalsentenceOutY
clear finalsentenceOutX
clear RNAAnswerX
clear RNAAnswerY
```

Código 3. CheckErrors.

```
%Script CheckWord
%This script find the index of the numeric word "palabraTemp" if this is
%part of the words file, and create it if does not exists.
%It creates the finalsentence variable, that will be the key to train the
RNA
flag=0;
tempindexword=0;
palabraTempLength=0;
numwords=numwords+1;
[rowwords,wordsLength]=size(words);
[empty,palabraTempLength]=size(palabraTemp);
palabraTemp=[palabraTemp,zeros(1,wordsLength-palabraTempLength)];
for j=1:rowwords %search in words file for the index of the word
    if palabraTemp(1,:)==words(j,:)
        if flag==0
            flag=1;
            tempindexword=j;
        end
    end
end
%If don't know this word,create new index
if flag==0
    flag=1;
    [rowwords,col2]=size(words);
    words(rowwords+1,:)=palabraTemp(1,:);
    tempindexword=rowwords+1;
end
finalsentence(m,numwords)=tempindexword;

if tempindexword==0
       error='Not tempindexword assigned'
end

palabraTemp=[];
```

Código 4. CheckWord.

```
%Script CheckWordOut
%This script find the index of the numeric word "palabraTemp" if this is
%part of the words file, and create it if does not exists.
%It creates the finalsentenceOut variable, that will be the key to train
the RNA
flag=0;
tempindexword=0;
palabraTempLength=0;
numwords=numwords+1;
[rowwords,wordsLength]=size(words2);
[empty,palabraTempLength]=size(palabraTemp);
palabraTemp=[palabraTemp,zeros(1,wordsLength-palabraTempLength)];
for j=1:rowwords %search in words file for the index of the word
    if palabraTemp(1,:)==words2(j,:)
        if flag==0
            flag=1;
            tempindexword=j;
        end
    end
end
%If don't know this word,create new index
if flag==0
    flag=1;
    [rowwords,col2]=size(words2);
    words2(rowwords+1,:)=palabraTemp(1,:);
    tempindexword=rowwords+1;

end
finalsentenceOut(m,numwords)=tempindexword;

if tempindexword==0
       error='Not tempindexword assigned'
end

palabraTemp=[];
```

Código 5. CheckWordOut.

```
%Script ConvWords
%The purpose of this script is just to convert the files words and words2
%to and easy legible format to the humans, to make easy the solving of
%posible problems and see the learning words rate.

wordsconverted=[];
wordsconverted2=[];
[rowwordscheck,wordsLengthcheck]=size(words);
[rowwordscheck2,wordsLengthcheck2]=size(words2);

for y=1:rowwordscheck
    for x=1:wordsLengthcheck
        wordsconverted=char(wordsconverted);
        wordsconverted(y,x)=words(y,x);
    end
end


for y=1:rowwordscheck2
    for x=1:wordsLengthcheck2
        wordsconverted2=char(wordsconverted2);
        wordsconverted2(y,x)=words2(y,x);
    end
end

clear rowwordscheck
clear rowwordscheck2
clear wordsLengthcheck
clear wordsLengthcheck2
clear y
clear x
```

Código 6. ConvWords.

```
%Script CrossValidation

finalsentenceReduced=finalsentence;
finalsentenceOutReduced=finalsentenceOut;

%RANDOM VECTOR CREATION
LengthRandomVector=length(in)/2-NumberOfTrain;
InputPair=length(finalsentenceReduced);
RandomVector = rand(LengthRandomVector,1,1);
RandomVector = RandomVector*(length(in)/2-2);
RandomVector = int16(RandomVector);
RandomVector=sortrows(RandomVector);

for k=1:length(RandomVector)
RandomVector(k,1)=RandomVector(k,1)+2;
end


%CREATING FINALSENTENCEREDUCED

for k=1:length(RandomVector)
finalsentenceReduced(:,(InputPair-(RandomVector(k,1))))=[];
end


%CREATING FINALSENTENCEOUTREDUCED

for k=1:length(RandomVector)
finalsentenceOutReduced(:,(InputPair-(RandomVector(k,1))))=[];
end

clear LengthRandomVector
clear k
clear InputPair
clear RandomVector
clear NumberOfTrain
```

Código 7. CrossValidation.

```
%Script InDimensions
%This script is just for take the variable"MaxLengthIn" number, it means
%the number of characters of the longest row. This is needed because the
%RNA will be created with this number of inputs. In the same way is
created
%the variable "MaxLengthOut" to know the number of outputs and use it in
%AlphaNumericConv script


load preload
%Check for correct input data

InputPair=length(in);
pairDetection()
if pair==0
    Error='Need a pairs of inputs/outputs'
end
clear pair;
clear n;
%MaxLengthOut creation with senar rows.
words=zeros(1,40);
words2=zeros(1,40);

words(1,1)=32;%The space is created.
words2(1,1)=32;

%Creation of ask file.

    if language==1
        ask=in;
        for a=length(in):-2:1
            ask(a,:)=[];
        end
    end

    if language==2
        ask=in;
        aux=length(in);
        aux=aux-1;
        for a=aux:-2:1
            ask(a,:)=[];
        end
        InvertOrder()
    end

%max length variables
MaxLengthOut=length(in{2,1});%This is the start of the defined row for
the
                             %first Out proposition.
if length(in)>1  %To assure enough input data
    for n=4:length(in);%The other Output sentences.
        if length(in)>n%To solve error of:Index exceeds matrix dimensions
            InputPair=n;
            pairDetection()
            if pair==0
                n=n+1;
            end
            if length(in{n,1})>MaxLengthOut;
                  MaxLengthOut=length(in{n,1});
            end
        end
```

```
    end
end

%MaxLengthIn
MaxLengthIn=length(in{1,1});
if length(in)>1  %To assure enough input data
    for n=3:length(in);
        if length(in)>n%To solve error of:Index exceeds matrix dimensions
            InputPair=n;
            pairDetection()
            if pair==1
                n=n+1;
            end
            if length(in{n,1})>MaxLengthIn;
                MaxLengthIn=length(in{n,1});
            end
        end
        n=n+1;
    end
end



clear InputPair;
clear pair;
clear n;
clear aux;
clear a;
Process='The InDimensions Function is passed correctly'
```

Código 8. InDimensions.

```
%Script to know if a number "InputPair" is pair or odd, with the variable
pair(1 if
%true)
%Called by InDimensions and AlphaNumericIn
aux1=char(InputPair);
aux2=int16(InputPair);
if aux1/2==aux2/2
    pair=1;
else
    pair=0;
end
clear aux1;
clear aux2;
```

Código 9. PairDetection.

```
%Script Reconversion
%This script needs the numeric variable "ReconvIn" and it will return the
variable
%"ReconvOut" as a Alphanumeric

ReconvOut=[];
TempConv=[];
TempConv=RNAAnswer';
[maxy,maxx]=size(TempConv);
[maxwords,empty]=size(words2);

%Is there any error?
for m=1:maxy
    for n=1:maxx
        if TempConv(m,n)>maxwords
            TempConv(m,n)=maxwords;
        end
    end
end

for n=1:maxy
    for m=1:maxx
            aux=TempConv(n,m);
            aux=words2(aux,:);
            aux=char(aux);
            ReconvOut{n,m}=aux;
    end
end
ReconvOut

%This variable will get all the sentences of the results of the questions

clear aux
clear n
clear m
clear TempConv
clear maxwords
clear empty
clear maxx
clear maxy
```

Código 10. Reconversion.

```
%Script RNA
%This script is the trainning and test of RNA

%Calculating number of neurons.
%[LongAlphaNumericIn,empty]=size(AlphaNumericIn);
%[LongAlphaNumericOut,empty]=size(AlphaNumericOut);
%[SavedLong,empty]=size(finalsentence);
[LongfinalsentenceOutReduced,empty]=size(finalsentenceOutReduced);
clear empty
%neur=SavedLong*LongAlphaNumericIn+LongFinalsentenceOut*LongAlphaNumericO
ut;
%neur=neur/LongFinalsentenceOut;
%neur=neur/2;
%savedneur=neur;
neur=250;

%Clear variables

clear aux
clear a

save PostLoad

clear MaxLengthIn
clear MaxLengthOut
clear Process
clear finalsentenceOut
clear ResAsk
clear SavedIn
clear SavedIn2
clear ask
clear in
clear language
clear words
clear words2
clear AlphaNumericIn
clear AlphaNumericOut
clear SavedLong
clear NumberOfTrain
clear FinalAutoLengthAdjustment
clear counter

net=newff([minmax(finalsentenceOutReduced)],[neur,LongfinalsentenceOutRed
uced],{'logsig','purelin'},'trainlm');
%net.trainParam.show = 1000;
%net.trainParam.lr = 0.05;
%net.trainParam.lr_inc = 1.05;
net.trainParam.mc = 0.05;
net.trainParam.mingrad= 0;
net.trainParam.epochs = 9000;
net.trainParam.time= 1000;

net.trainParam.goal = 0;
net.layers{1}.initFcn = 'initwb';
%net.layers{:,:}.initFcn = 'initwb';
net.inputWeights{1,1}.initFcn = 'rands';
%net.biases{:,:}.initFcn = 'rands';
net.biases{1,1}.initFcn = 'rands';
net.biases{2,1}.initFcn = 'rands';
net = init(net);
```

```
net=train(net,finalsentenceReduced,finalsentenceOutReduced);
a = sim(net,finalsentence);
RNAAnswer=a;
RNAAnswer=int16(RNAAnswer);
RNAAnswer=RNAAnswer';
Process='The RNA Function is passed correctly'
load PostLoad
save RNA
clear a
clear neur
clear LongfinalsentenceOutReduced
clear finalsentenceReduced
clear finalsentence finalsentenceOutReduced
```

Código 11. RNA.

```
%Script RNALongSelection()
%It is called by main().
%This script is the trainning and test of the lenght of RNA



%Preparing the Out size variable.
GetCounterOut()

%Calculating number of neurons.
neur=50;


%Clear variables

%clear MaxLengthIn
%clear MaxLengthOut
%clear Process
%clear finalsentence
%clear finalsentenceOut
%clear ResAsk
%clear SavedIn
%clear SavedIn2
%clear ask
%clear in
%clear words
%clear words2
%clear AlphaNumericIn
%clear AlphaNumericOut
%clear SavedLong
%clear NumberOfTrain
%clear finalsentenceOutReduced

net=newff([minmax(finalsentenceReduced)],[neur,1],{'logsig','purelin'},'t
rainlm');
%net.trainParam.show = 1000;
%net.trainParam.lr = 0.05;
%net.trainParam.lr_inc = 1.05;
net.trainParam.mc = 0.05;
net.trainParam.mingrad= 0;
net.trainParam.epochs = 9000;
net.trainParam.time= 60;

net.trainParam.goal = 0;
net.layers{1}.initFcn = 'initwb';
%net.layers{:,:}.initFcn = 'initwb';
net.inputWeights{1,1}.initFcn = 'rands';
%net.biases{:,:}.initFcn = 'rands';
net.biases{1,1}.initFcn = 'rands';
net.biases{2,1}.initFcn = 'rands';
net = init(net);

net=train(net,finalsentenceReduced,CounterOut);



a = sim(net,finalsentence);
FinalAutoLengthAdjustment=int16(a);

Process='The RNA Function is passed correctly'
```

```
clear CounterOut
clear a
clear net

Código 12. RNALongSelection()
```

```
%Script WordSeparation


%WordSeparation of the input text

palabraTemp=[];
[row,col]=size(AlphaNumericIn);
[row2,col2]=size(words);
numwords=0;
finalsentence=[];
for m=1:row %The rows of the input text
    numwords=0;
    for longword=1:col %The columns of a row
        if AlphaNumericIn(m,longword)~=0
            if
AlphaNumericIn(m,longword)==32|AlphaNumericIn(m,longword)==46|AlphaNumeri
cIn(m,longword)==44 %When a space, point or coma is founded.
                CheckWord()
            else
                palabraTemp=[palabraTemp,AlphaNumericIn(m,longword)];
            end
        end
    end
end

clear col
clear col2
clear empty
clear row
clear row2
clear m
clear j

clear tempindexword
clear wordsLength
clear rowwords
clear palabraTempLength
clear palabraTemp
clear numwords
clear longword
clear flag

%WordSeparation of the output text

palabraTemp=[];
[row,col]=size(AlphaNumericOut);
[row2,col2]=size(words);
numwords=0;
m=0;
for m=1:row %The rows of the input text
    numwords=0;
    for longword=1:col %The columns of a row
        if AlphaNumericOut(m,longword)~=0
            if
AlphaNumericOut(m,longword)==32|AlphaNumericOut(m,longword)==46|AlphaNume
ricOut(m,longword)==44 %When a space is founded
                CheckWordOut()
            else
                palabraTemp=[palabraTemp,AlphaNumericOut(m,longword)];
            end
        end
```

```
    end
end

clear col
clear col2
clear empty
clear flag
clear row
clear row2
clear rowwords
clear palabraTemp
clear palabraTempLength
clear tempindexword
clear wordsLength
clear m
clear numwords
clear j
clear longword
clear TempWord

finalsentenceOut=finalsentenceOut';
finalsentence=finalsentence';

[finalsentencey,finalsentencex]=size(finalsentence);
for y=1:finalsentencey
    for x=1:finalsentencex
        if finalsentence(y,x)==0
            finalsentence(y,x)=1;
        end
    end
end

for y=1:finalsentencey
    for x=1:finalsentencex
        if finalsentenceOut(y,x)==0
            finalsentenceOut(y,x)=1;
        end
    end
end


clear finalsentencex
clear x
clear y
clear AlphaNumericOut
clear AlphaNumericIn

%Process='The WordSeparation Function is passed correctly'
```

Código 13. WordSeparation.

```
%Script DeleteExcedingWords()
%This script is is used to delete the exceding words of the variable
%RESASk, with the array vector made by the secondary RNA.


[yRNAAnswer,xRNAAnswer]=size(RNAAnswer);
[yFinalAutoLengthAdjustment,xFinalAutoLengthAdjustment]=size(FinalAutoLen
gthAdjustment);
[maxwords,empty]=size(words2);

n=0;
%Deleting
for y=1:yRNAAnswer;
    for x=1:xRNAAnswer;
        if FinalAutoLengthAdjustment(1,y)<x;
           RNAAnswer(y,x)=1;

        end
    end
end

%Is there any error?
for m=1:yRNAAnswer;
    for n=1:xRNAAnswer
        if RNAAnswer(m,n)>maxwords
            ErrorCode=': The RNA was not able to get succesful Output'
        end
    end
end

%Special features to do the conflictive numbers, a chance to have succes


for n=1:yRNAAnswer
    for m=1:xRNAAnswer
    %if TempConv(n,m)~=0
            if RNAAnswer(n,m)<=0
                RNAAnswer(n,m)=1;%A space is defined
              %     WarnningCode=':   The   RNA   could   give   innacurate
results(negative results)'
            end

            aux=RNAAnswer(n,m);
            if aux>maxwords
                WarnningCode=':    The    RNA    could    give    innacurate
results(high results)'
                aux=maxwords;%Aproximation to max indexed word
            end
    end
end


clear xFinalAutoLengthAdjustment
clear yFinalAutoLengthAdjustment
clear maxx
clear maxy
clear maxwords
clear empty
clear yRNAAnswer
clear xRNAAnswer
clear n
```

```
clear m
clear x
clear y
clear aux
```

Código 14. DeleteExcedingWords.

```
%Script GetCounterOut()
%It is called by RNALongSelection().

%The variable CounterOut will be an array, with the long of the
%input/output text,and will be used to delete exceding words of the final
%response.

CounterOut=[];
counter=0;
[mcounter,ncounter]=size(finalsentenceOutReduced);
for n=1:ncounter;
    for m=1:mcounter;
        if finalsentenceOutReduced(m,n)>1
            counter=counter+1;
        end
    end
    CounterOut(1,n)=counter;
    counter=0;
end

clear counter
clear m
clear mcounter
clear ncounter
clear n
```

Código 15. GetCounterOut

```
%Script InvertOrder
%This script has the purpose of switch the pair and nopair sentences.

%duplicate the sentences
for a=1:2:length(in)
    in(a+1,:)=in(a,:);
end

%overwrite sentences
for a=1:2:length(in)
    in(a,:)=ask((a+1)/2,:);
end

clear ask
```

Código 16. InvertOrder.

7.2.2 Códigos main2()

```
%Main2()
%Second program to be executed if want to do an specific question.

 %If the program one has not been executated.
%load the variable ask like that example: ask='yo canto.'This for
%language=2 %if the ask is in the secondary language
language=1 %if the ask is in the secondary language

    if  language==1%If  it's  selected,  it  means  is  in  the  primary
language(english in this case) the ask file

        load finalnet
        ask='I sing a beatiful chair.';
    end
    if  language==2%If  it's  selected,  it  means  is  in  the  primary
language(english in this case) the ask file
        ask='yo canto una silla bonita.';
        load finalnet2
    end

WordSeparationAsk()
in=ones(1,finalsentencey);

[empty,finalsentenceX]=size(finalsentence);
for n=1:finalsentenceX
     in(1,n)=finalsentence(1,n);
end

clear finalsenteneX
clear finalsentence
clear n
clear empty
in=in';
RNAAnswer = sim(net,in);
clear in
RNAAnswer= int16(RNAAnswer);

DeleteExcedingWords()%Returns RNAAnswer (upgraded)
Reconversion()
```

Código 16. main2.

```
%Script WordSeparationAsk


palabraTemp=[];
[yin,xin]=size(ask);
[ywords,xwords]=size(words);
numwords=0;
finalsentence=[];

for longword=1:xin %The columns of a row
    if ask(1,longword)~=0
        if    ask(1,longword)==32|ask(1,longword)==46|ask(1,longword)==44
%When a space, point or coma is founded.
             CheckWordAsk()
        else
             palabraTemp=[palabraTemp,ask(1,longword)];
        end
    end
end


clear yin
clear xin
clear empty
clear yin
clear xin
clear m
clear j

clear tempindexword
clear wordsLength
clear rowwords
clear palabraTempLength
clear palabraTemp
clear numwords
clear longword
clear flag
clear ywords2
clear xwords2

%Process='The WordSeparation Function is passed correctly'
```

Código 17. WordSeparationAsk.

```
%Script CheckWordask
%This script find the index of the numeric word "palabraTemp" if this is
%part of the words file, and create it if does not exists.
%It creates the finalsentence variable, that will be the key to train the
RNA
flag=0;
tempindexword=0;
palabraTempLength=0;
numwords=numwords+1;
[ywords,wordsLength]=size(words);
[empty,palabraTempLength]=size(palabraTemp);
palabraTemp=[palabraTemp,zeros(1,wordsLength-palabraTempLength)];
for j=1:ywords %search in words file for the index of the word
    if palabraTemp(1,:)==words(j,:)
        if flag==0
            flag=1;
            tempindexword=j;
        end
    end
end
%If don't know this word,create new index
if flag==0
    flag=1;
    [ywords,xwords]=size(words);
    words(ywords+1,:)=palabraTemp(1,:);
    tempindexword=ywords+1;
    Warnning='New word created in ask step,the new word is:'
    palabraTemp(1,:)

end
finalsentence(1,numwords)=tempindexword;

if tempindexword==0
       error='Not tempindexword assigned'
end

palabraTemp=[];
```

Código 18. CheckWordask.

Traducción automatizada basada en algoritmos de redes neuronales artificiales

REFERENCIAS

Titulación: Enginyeria Tècnica Industrial en Electrònica Industrial

AUTOR: Xavier Escobar
DIRECTORES: Jesús Brezmes, Eduard Llobet

Fecha: Junio / 2009 .

8.- Referencias

[1] José *Ranilla*, Centro de *Inteligencia Artificial*. Universidad de Oviedo en Gijón. Campus de Viesques.

[2] "ELIZA - Un Programa informático para el estudio del lenguaje natural en la comunicación Hombre-Máquina", *Communications of the Association for Computing Machinery 9* (1966): 36-45.

[3] ASCII. - El código ASCII (acrónimo inglés de American Standard Code for Information Interchange — (Código Estadounidense Estándar para el Intercambio de Información), pronunciado generalmente [áski], es un código de caracteres basado en el alfabeto latino tal como se usa en inglés moderno y en otras lenguas occidentales. Fue creado en 1963 por el Comité Estadounidense de Estándares (ASA, conocido desde 1969 como el Instituto Estadounidense de Estándares Nacionales, o ANSI) como una refundición o evolución de los conjuntos de códigos utilizados entonces en telegrafía. Más tarde, en 1967, se incluyeron las minúsculas, y se redefinieron algunos códigos de control para formar el código conocido como US-ASCII.
El código ASCII utiliza 8 bits para representar los caracteres, aunque inicialmente empleaba un bit adicional (bit de paridad) que se usaba para detectar errores en la transmisión. A menudo se llama incorrectamente ASCII a otros códigos de caracteres de 8 bits, como el estándar ISO-8859-1 que es una extensión que utiliza 8 bits para proporcionar caracteres adicionales usados en idiomas distintos al inglés, como el español.
ASCII fue publicado como estándar por primera vez en 1967 y fue actualizado por última vez en 1986. En la actualidad define códigos para 33 caracteres no imprimibles, de los cuales la mayoría son caracteres de control obsoletos que tienen efecto sobre como se procesa el texto, más otros 95 caracteres imprimibles que les siguen en la numeración (empezando por el carácter espacio).
Casi todos los sistemas informáticos actuales utilizan el código ASCII o una extensión compatible para representar textos y para el control de dispositivos que manejan texto.