42 JAIIO EST 2013 16° Concurso de Trabajos Estudiantiles

Vail: Un Framework para el Desarrollo de Aplicaciones con Redes Neuronales Artificiales

Autores:

María Julia Blas – mariajuliablas@gmail.com
Nicolás Emilio Díaz Ferreyra – nicoediaz@gmail.com
Juan Leonardo Sarli – juanleonardosarli@gmail.com

Institución:

UTN Regional Santa Fe

Carrera:

Ingeniería en Sistemas de Información

Cátedra:

Inteligencia Computacional

Docentes:

Dra. Georgina Stegmayer
Dra. María de los Milagros Gutierrez

Vail: Un Framework para el Desarrollo de Aplicaciones con Redes Neuronales Artificiales

Abstract. El presente trabajo fue desarrollado en el marco de la asignatura Inteligencia Computacional de la UTN-FRSF a lo largo del 2º cuatrimestre del ciclo lectivo 2012. Presenta la implementación de un aplicativo llamado Vail, el cual sirve como plataforma para modelar soluciones basadas en el uso de Redes Neuronales Artificiales. Los fundamentos teóricos que enmarcaron su origen, junto con la descripción de su desarrollo y posterior organización en paquetes y clases; son algunos de los principales temas que se presentan. Se incluye además, la resolución de un caso práctico que utiliza como soporte de desarrollo el framework implementado. De esta manera, se muestran las ventajas y desventajas del mismo, obteniéndose una herramienta que podría ser utilizada con fines educacionales para la enseñanza de Redes Neuronales.

Keywords. Inteligencia computacional, redes neuronales artificiales, framework, reconocimiento de caracteres, código morse.

1 Introducción

En la actualidad, las ciencias de la computación y los sistemas de información están abocados a la resolución de problemas de alta complejidad, en donde pueden presentarse situaciones en las cuales se desconozca completamente el sistema a ser analizado o las condiciones del ambiente estén sujetas a cambios permanentes y desconocidos. Es por esto que resulta imprescindible el desarrollo de aplicaciones, técnicas y herramientas que brinden solución a este tipo problemas; en los cuales, los métodos de resolución clásicos son difíciles de implementar.

La Inteligencia Computacional es una rama de la Inteligencia Artificial centrada en el estudio de mecanismos adaptativos que permiten el comportamiento inteligente de sistemas complejos y cambiantes [1]. Combina elementos de aprendizaje, adaptación y evolución para crear programas que son, de cierta forma, inteligentes. La aplicación de técnicas de redes neuronales, lógica difusa, algoritmos genéticos y sistemas expertos, permite identificar patrones complejos en conjuntos de datos, tomar decisiones basadas en juicios cualitativos y cuantitativos, introducir reglas de decisión a partir de datos históricos y encontrar soluciones a problemas con alta complejidad de variables.

Una Red Neuronal Artificial (RNA) es un conjunto de procesadores elementales vinculados o neuronas artificiales interconectadas, que realizan algunas de las siguientes funciones: aprendizaje, memorización, generalización o abstracción de características esenciales [2]. Una RNA plantea un modelo de computación que se asemeja más a la fisiología del cerebro humano que al modelo de computación tradicional [3]. El comportamiento de una neurona artificial es similar al de una neurona biológica, tanto en lo relativo a su funcionamiento como a su interconexión. Las RNA permiten obtener resultados razonables en problemas donde las entradas presentan ruido, aleatorie-

dad o se encuentran incompletas. Entre sus áreas de aplicación se destacan, entre otras, conversión de texto a voz, procesamiento del lenguaje natural, reconocimiento de caracteres e imágenes, control de procesos y robótica [4-9].

El desarrollo de sistemas basados en RNA implica seguir y respetar un conjunto de lineamientos impuestos por esta disciplina. Dada la existencia de una amplia gama de aplicaciones, resulta de interés contar con un marco de trabajo que sirva como guía durante el proceso de desarrollo de tales sistemas. En la actualidad existen diversos aplicativos orientados a esta temática, entre los cuales se destacan Neuroph[1], AForge.NET[2] y FANN[3]. Sin embargo, estos suelen ser demasiado complejos para un usuario inexperto en el área. Otra alternativa es utilizar lenguajes de programación para la codificación de RNA. Lenguajes como Matlab®[4] resultan altamente potentes, pero como contrapartida requieren de un alto nivel de conocimiento de los modelos matemáticos que fundamentan dichas redes.

En este contexto, este trabajo propone un framework en un lenguaje de programación orientado a objetos provisto de un conjunto de clases y métodos que sirven como base para la organización y desarrollo de sistemas basados en RNA. Debido a que las neuronas biológicas utilizan para comunicarse mensajes codificados en lo que podría asemejarse a un "Código Morse Neuronal" [10], se decidió nombrar "Vail" a la implementación Java de esta propuesta; en honor a Alfred Vail, quien colaboró en el año 1880 en la creación del código Morse.

En las siguientes secciones se detalla una breve descripción de los conceptos relacionados con RNA, la implementación propuesta y la resolución de un caso práctico junto con un análisis de los resultados obtenidos; concluyendo, de esta manera, sobre las ventajas y desventajas del aplicativo desarrollado.

2 Redes Neuronales Artificiales

Existen diferentes topologías de RNA. Una de las más utilizadas es el Perceptrón Multicapa (MLP), el cual consta de una capa de neuronas de entrada, al menos una capa de neuronas ocultas y una capa de neuronas de salida (Figura 1). Dicha topología es capaz de establecer una relación entre dos conjuntos de datos por medio de la propagación hacia delante de la señal de entrada a través de los pesos sinápticos.

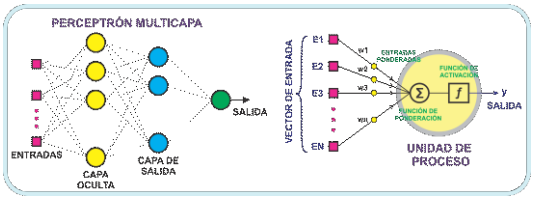

Fig. 1. Componentes de una RNA.

[1] http://neuroph.sourceforge.net/about_project.html
[2] http://www.aforgenet.com/framework/features/neural_networks.html
[3] http://leenissen.dk/fann/wp/
[4] http://www.mathworks.com/products/matlab/

Previo a la utilización de una RNA se requiere de la aplicación de un algoritmo de aprendizaje que optimice dichos pesos en función del error cometido, como por ej. Backpropagation. Sin embargo, puede ocurrir que durante el entrenamiento la red pierda la capacidad de generalización (es decir, "memorice" los ejemplos que se le presentan). Éste fenómeno se lo conoce como Overfitting, y puede solucionarse utilizando criterios de terminación temprana; como por ej. Early Stopping. Este criterio, detiene el proceso de aprendizaje cuando el error de validación comienza a aumentar.

3 Desarrollo del Framework

En ingeniería de software, un framework es una estructura de soporte en base a la cual un proyecto de software puede ser organizado y desarrollado. Como se mencionó anteriormente, la mayoría de los frameworks existentes no se encuentran orientados a la enseñanza de arquitecturas con RNA. Debido a que la semántica de los lenguajes orientados a objetos da lugar a un manejo más intuitivo de los conceptos, se decidió implementar Vail en Java.

3.1 Definición de la Arquitectura

El paradigma orientado a objetos intenta imitar el pensamiento humano mediante la utilización de abstracciones que representan las distintas entidades del dominio de un problema. Junto con estas abstracciones define un conjunto de relaciones permitidas, entre las que se destacan:

- *Relación de Asociación*: Permite definir interacción entre clases de objetos.
- *Relación de Todo/Parte*: Permite definir asociaciones y composiciones entre clases de objetos.
- *Relación de Generalización/Especialización*: Permite definir herencia entre clases de objetos.

Tomando estos lineamientos como punto de partida, y con el objetivo de definir una estructura que refleje las relaciones entre conceptos, se decidió implementar un conjunto de paquetes que represente los posibles componentes a utilizar en una implementación de RNA. La Figura 2, esquematiza la división planteada.

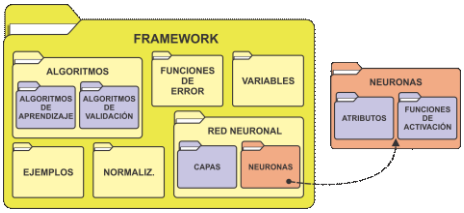

Fig. 2. Arquitectura de Vail

Cada paquete contiene las clases asociadas al concepto que representa, lo que facilita la identificación de los componentes al momento de definir una solución. Ade-

más, con el objetivo de lograr adaptabilidad, en cada uno de ellos se incorporaron puntos de flexibilización (clases que pueden ser extendidas para resolver un problema concreto). De esta manera, se logró una arquitectura simple e intuitiva; en la cual, el entendimiento de las relaciones entre clases es fácil de asimilar.

3.2 Contenido de los Paquetes Principales

utn.frsf.isi.ic.framework.redneuronal.
Contiene la clase abstracta *RedNeuronal* que define todos los atributos propios de una red neuronal genérica. Incluye además la clase *MLP*, que extiende de la clase anterior con el objetivo de especificar una red del tipo Perceptrón Multicapa [11].

utn.frsf.isi.ic.framework.redneuronal.neuronas.
Contiene la abstracción *Neurona* que define el comportamiento genérico del componente, dando así un punto de flexibilización. Incluye las implementaciones *PerceptronSimple* y *NeuronaRadial*, que se usan en especializaciones de *RedNeuronal*.

utn.frsf.isi.ic.framework.algoritmos.algoritmosaprendizaje.
Contiene la clase abstracta *AlgoritmoDeEntrenamiento* la cual posee como atributos la red neuronal, la función de error y los ejemplos de entrenamiento a ser utilizados. Además define un método abstracto *entrenamiento()* que debe ser implementado en los distintos algoritmos, ya que refiere específicamente al proceso de aprendizaje. Incluye dos implementaciones (clases) del algoritmo de propagación hacia atrás [11]:

- *Backpropagation*: Finaliza luego de transcurrida una determinada cantidad de épocas o cuando el error ha disminuido a un valor previamente especificado.
- *BackpropagationConEarlyStopping*: Similar a *Backpropagation* pero incorpora un criterio de terminación. Evita que se cometa over-fitting, deteniendo su ejecución cuando el error de validación aumenta mientras el de entrenamiento disminuye.

utn.frsf.isi.ic.framework.algoritmos.algoritmosvalidacion.
Contiene dos clases: *Fold* y *KFoldCrossValidation*. La primera, es una clase abstracta que define un punto de flexibilización; ya que para aplicar el algoritmo K-Fold Cross Validation [1] el usuario debe crear su propio fold extendiendo dicha clase e implementando los métodos correspondientes al entrenamiento y testeo del mismo. Por otro lado, la clase *KFoldCrossValidation* aplica el algoritmo de validación sobre una lista de folds, y luego informa la tasa de reconocimiento global.

utn.frsf.isi.ic.framework.normalizaciones.
Contiene una clase abstracta *Normalización* a partir de la cual pueden generarse los distintos procesos de normalización de las variables. Incluye 2 implementaciones (clases): *NormalizaciónGlobal* y *NormalizaciónEnRango*.

utn.frsf.isi.ic.framework.variables.
Contiene la clase abstracta *Vector* que modela un conjunto de variables de entrada/salida genérico. Incluye la definición concreta de los tipos de vectores clásicos: *VectorHeterogeneo*, *VectorHomogeneo* y *VectorTopológico*.

3.3 Otros Paquetes

Como complemento a la definición de los conceptos principales, se construyó un conjunto de paquetes que sirve como apoyo al proceso de desarrollo. *Ejemplos* y *Funcioneserror*, contienen clases que modelan abstracciones que son utilizadas en los algoritmos; mientras que *Neuronas.Funcionesactivacion* y *Capas* incluyen implementaciones que ayudan en la configuración de las extensiones de *RedNeuronal*.

3.4 Creación de una Librería

Con el objetivo de aportar a la comunidad de desarrolladores, el framework se encapsuló en una librería (.jar). Las clases se encuentran documentadas al estilo JavaDoc e implementan la interfaz Serializable (lo que facilita su almacenamiento).

4 Caso Práctico: Reconocimiento de Caracteres Morse

El Código Morse (CM) fue desarrollado en 1834 por Alfred Vail mientras colaboraba con Samuel Morse en la invención del telégrafo eléctrico. Vail creó un mecanismo de comunicación en el cual cada letra del alfabeto y cada dígito numérico, era transformado individualmente según un código consistente de hasta cinco rayas y puntos; el cual se lograba con un conjunto de señales telegráficas cuya diferencia radicaba en el tiempo de duración de la señal activa (Figura 3). La duración de un punto era la mínima posible, mientras que la de una raya era aproximadamente el equivalente a tres puntos. Entre cada par de símbolos de un mismo carácter, se daba una ausencia de señal de duración similar a un punto. Caracteres consecutivos de una misma palabra y distintas palabras de una misma oración, se diferenciaban por la falta de señal durante un período equivalente a tres puntos y tres rayas, respectivamente. Morse reconoció la idoneidad de este sistema y lo patentó junto con el telégrafo eléctrico, siendo conocido como “American Morse Code” [12].

A lo largo de la historia, el CM ha sido de gran utilidad. Fue empleado como mecanismo de comunicación en las líneas telegráficas y en las transmisiones por radio, hasta que entró en desuso debido al desarrollo de otras tecnologías [13]. De esta manera, se convirtió en una pieza fundamental de las comunicaciones marítimas y aéreas, siendo vigente su utilización en aviación instrumental.

Fig. 3. Caracteres en Código Morse.

Diversos autores han aplicado RNA en problemas relativos al reconocimiento de patrones y de clases [14-15]. Un subconjunto de estos trabajos, propone la utilización de este tipo de modelos como intérpretes de un código, reconociendo un caracter real en base a un caracter codificado y/o viceversa [16-19].

En este contexto, puede afirmarse que el reconocimiento de caracteres Morse presenta un conflicto cuya solución puede plantearse con RNA. Tomando este problema como caso práctico, se utilizará el framework implementado para codificar una solución con el objetivo de demostrar las capacidades de adaptabilidad, confiabilidad, usabilidad y robustez, entre otras, de la librería desarrollada.

4.1 Descripción del Problema

La velocidad de transmisión de los caracteres y de las palabras que forman el texto de un mensaje Morse depende en gran medida de la habilidad y experiencia que tenga el telegrafista, tanto a la hora de transmitir como de recibir los mensajes. Esto se debe a que diferentes emisores presentarán distintos patrones de duración para un punto y/o raya; junto con una pequeña desviación propia de las duraciones de los caracteres.

Sean P1 y P2 dos personas que desean emitir un mensaje Morse que contenga el carácter "Q" mediante un telégrafo, y sea P3 el receptor de este mensaje. Siendo (150,151,51,153) y (300,305,99,295) las duraciones de los pulsos telegráficos de P1 y P2 respectivamente, ambos conjuntos respetan las proporciones requeridas entre duraciones de puntos y rayas. Sin embargo, aunque representan el mismo carácter es claro que no son iguales. Dependerá de la experticia de P3, el reconocer que un punto del individuo P1 toma aproximadamente una duración de 50, mientras que para el individuo P2 toma un valor cercano a 100; y de esta manera poder detectar cuales duraciones corresponden a puntos y cuales a rayas.

Pero ésta no es la única variación que debe afrontar el receptor, ya que la interpretación del mensaje puede verse afectada por la variabilidad proveniente de la naturaleza humana del emisor. En el ejemplo, P1 ha señalado que {150, 151, 153} pertenecen al conjunto raya, y P2 ha hecho lo mismo con {300, 305, 295}. Es evidente, que por ser humanos es prácticamente imposible que P1 y P2 eviten cierta desviación entre las duraciones de un mismo símbolo. Es más, probablemente una retransmisión del mensaje no coincida en ningún valor con el mensaje original. En este caso, nuevamente dependerá de la experticia de P3 el comprender que existen desviaciones que son imposibles de eliminar y que afectarán, en mayor o menor medida, el entendimiento del mensaje final.

De esta manera, el objetivo del caso práctico propuesto es reconocer caracteres Morse, tomando en consideración las variabilidades asociadas al emisor, en base a un conjunto de valores de entrada asociados a duraciones.

4.2 Estrategia de Solución e Implementación con el Framework

Variables de Entrada.
Un caracter Morse queda definido por un conjunto de hasta cinco pulsos eléctricos. Dada la necesidad de medir la duración de este tipo de pulsos y en ausencia de un telégrafo, a los fines prácticos de lograr medir duraciones, se decidió trabajar en base a pulsos de mouse (clicks).

Se definió un vector de variables de entrada de 5 componentes, en el cual la i-ésima posición corresponde a la duración en milisegundos del i-ésimo pulso del ca-

racter. En el caso de ser un caracter con una cantidad de pulsos inferior a 5, las componentes inexistentes valen 0. De esta manera, el conjunto de variables de entrada conforma un vector homogéneo. La implementación de este vector, se realizó extendiendo la clase *VectorHomogeneo* sobre una nueva clase *VariablesDeEntrada* en la cual se especificaron los métodos referidos a su comportamiento.

Normalización.
Dada la variabilidad existente en las duraciones de los clicks, se planteó la necesidad de establecer un mecanismo que permita que diferentes patrones tengan importancia similar dentro de la red. Esto conllevó a la decisión de incorporar un proceso de normalización sobre los datos de entrada.

Debido a que ciertos caracteres Morse quedan conformados por un mismo tipo de símbolo, la utilización de normalización global estricta no es factible. Esto se debe a que la importancia de un símbolo es relativa a la existencia del otro; por lo que en los casos donde existe un único símbolo no puede establecerse una relación.

Por tal motivo se decidió implementar un proceso de normalización global adaptado. Éste, en lugar de trabajar con el máximo valor del patrón ingresado, normaliza cada una de las entradas en base a un valor fijo llamado "normalizador". Durante el entrenamiento el máximo valor existente en el conjunto de ejemplos es tomado como "normalizador"; mientras que durante la utilización de la RNA se toma como valor el promedio de un conjunto de muestras emitidas por el usuario. Esto último permite adaptar la aplicación a usuarios con diferentes patrones.

De esta manera se logra que datos que representan un mismo caracter, pero que han sido ingresados por diferentes emisores, tengan un nivel de influencia similar sobre el modelo. Además, se garantiza que la representación de caracteres formados por un único tipo de símbolo no generará confusiones. La implementación de este proceso de normalización se realizó creando una nueva clase llamada *NormalizacionGlobalTotal*. Ésta, extiende de la clase *NormalizacionGlobal*, y es utilizada únicamente para redefinir el valor que debe tomarse como máximo.

Definición de Topologías.
Debido a que el caso práctico es un problema de clasificación, se optó por resolverlo en base a dos topologías MLP: T1 y T2. En ambos casos, la cantidad de neuronas a colocar en la capa oculta se determinó de forma experimental. T1 se definió como un MLP con 10 neuronas en la capa oculta y 36 en la capa de salida (una por cada caracter a reconocer). Su implementación se realizó instanciando la clase *MLP* del framework. T2 se definió como un conjunto de redes MLP, en donde cada una de ellas reconoce una clase. Cada red de este modelo contiene 1 neurona de salida y 2 neuronas en la capa oculta. Su implementación se llevó a cabo codificando una nueva clase llamada *ConjuntoDeMLP*, la cual internamente genera un conjunto de instancias de la clase *MLP*. En ambos casos, las neuronas utilizaron la función de activación sigmoide provista por el framework (clase *FuncionSigmoide*).

La utilización de ambos topologías conllevó a un análisis de resultados y a una posterior evaluación del comportamiento de cada una de ellas.

Entrenamiento, Validación y Prueba.
El entrenamiento de los modelos neuronales se realizó instanciando la clase *BackpropagationConEarlyStopping*. Con el objetivo de lograr una alta tasa de clasificación, se estimaron los parámetros mediante análisis experimental. En ambos casos se seteó la cantidad de épocas en 15000, el error en 0.003 y la velocidad de aprendizaje en 0.05. Además se instanció la clase *KFoldCroossValidation* con el objetivo de realizar una validación cruzada entre los datos de entrenamiento y prueba. Se extendió la clase *Fold*, creándose *FoldRedCM* y *FoldConjuntoCM*; lo que permitió redefinir los métodos necesarios para el entrenamiento y la validación de cada topología.

La división de ejemplos se realizó dividiendo la totalidad de ejemplos de cada caracter (40) en 4 grupos; en donde el i-ésimo grupo corresponde al conjunto de prueba del i-ésimo fold. Una segunda partición del conjunto de entrenamiento (20%) se utilizó en el proceso de Early-Stopping.

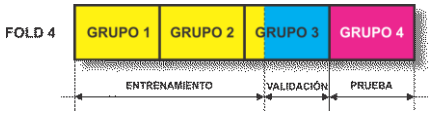

Fig. 4. División de ejemplos para el fold 4.

Creación de Ejemplos.
Dada la inexistencia de datos de entrenamiento en relación al problema planteado, los mismos debieron ser generados. Esto conllevó al desarrollo de un mecanismo de soporte para su captura y posterior almacenamiento. Su implementación se realizó instanciando la clase *EjemploEntrenamiento* del framework.

4.3 Extensiones Realizadas al Framework

La Figura 5 muestra un subconjunto de clases implementadas para dar solución al problema planteado. En ella, se esquematizan las clases del framework junto con las extensiones realizadas mediante la utilización de los puntos de flexibilización.

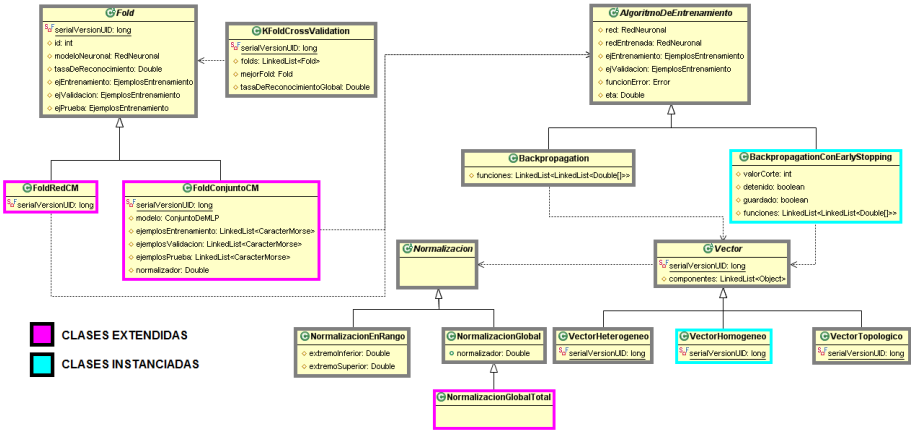

Fig. 5. Subconjunto de clases utilizadas.

4.4 Resultados

Finalizado el entrenamiento y la validación, se obtuvieron los siguientes resultados.

En T1 la tasa de clasificación lograda fue de 96.18%, habiéndose detenido el proceso de entrenamiento por error. Un análisis de la matriz de confusión (matriz en la que cada elemento m[i][j], indica la cantidad de ejemplos que debían ser clasificados como “j”, y fueron clasificados como “i”), reveló que la red tiende a confundirse caracteres en los que existe un símbolo de diferencia. Por ejemplo, en algunos casos confunde “E” (punto) con “T” (raya). Esto se debe a que el punto de la letra E ha sido ingresado con una duración similar a una raya. En T2, la tasa de clasificación fue de 92.27%. De las 36 redes, 25 detuvieron su aprendizaje por épocas, mientras que las 9 restantes lo hicieron por error. La matriz de confusión obtenida reveló un comportamiento similar al de T1 en lo referente a caracteres Morse que difieren en un símbolo.

De esta manera, puede concluirse que ambos modelos presentaron resultados satisfactorios (dada la alta tasa de clasificación obtenida), siendo T1 la topología que mejor se ajusta al caso práctico planteado. Sin embargo, no debe perderse de vista que en el correcto funcionamiento del modelo es clave que las rayas tengan una duración más prolongada que los puntos.

5 Conclusiones

Los conocimientos teóricos sobre RNA y su aplicación a problemas de ingeniería, suponen un alto grado de experticia en esta temática. Es por esto que en un contexto educacional, la existencia de una herramienta que facilite la manipulación de estos conceptos es extremadamente útil.

La propuesta presentada, destaca las características relevantes del trabajo con RNA; siendo de utilidad como guía durante el proceso de aprendizaje. Se ha demostrado su aplicación a un caso práctico concreto, en el cual es posible comprender la conformación de un modelo neuronal junto con la especificación del problema. El framework desarrollado ha brindado un marco de trabajo simple e intuitivo, sobre el cual el desarrollo del modelo conceptual junto con la definición del comportamiento específico requerido para el problema se ha realizado de forma rápida y ágil. Esto se debió a que la lógica de las RNA y de los algoritmos asociados a su entrenamiento y validación, forma parte de las clases disponibles; por lo que, con una simple instanciación de las características deseadas se obtuvo un aplicativo funcional. La adaptabilidad ofrecida en relación a los puntos de flexibilización facilitó la extensión de comportamientos particulares en los casos específicos, como por ejemplo durante el proceso de normalización. Aunque la evaluación de resultados específicos (como por ejemplo la matriz de confusión) no forma parte del framework, su codificación puede realizarse de manera sencilla tomando como base la información que contienen las clases del aplicativo.

De esta manera, Vail ha cumplido con los objetivos planteados, contribuyendo al desarrollo de aplicaciones como una colección de clases (abstractas y concretas) y métodos orientados a abordar la resolución de problemas, presentando versatilidad, robustez y utilidad en relación al trabajo con RNA.

Referencias

1. Engelbrecht, A.: Computational Intelligence: An Introduction. John Wiley & Sons (2007).
2. Gomez, F., Fernandez, M., López, M., Diaz, M.: Aprendizaje con Redes Neuronales Artificiales. Dpto de Informática, Escuela Universitaria Politécnica de Albacete, Universidad de Castilla-La Mancha (2008). Url: http://www.uclm.es/ab/educacion/ensayos/pdf/revista9/9_19.pdf
3. Romero, L., Calonge, T.: Redes Neuronales y Reconocimiento de Patrones. Universidad de Salamanca, España (2001). Url: http://gredos.usal.es/jspui/bitstream/10366/55893/1/DIA_Redes%20neuronales.pdf
4. Olabe, X.: Redes Neuronales Artificiales y sus Aplicaciones. Escuela Superior de Ingeniería de Bilbao, EHU (2009). Url: http://cvb.ehu.es/open_course_ware/castellano/tecnicas/redes_neuro/contenidos/pdf/libro-del-curso.pdf
5. Karaali, O., Corrigan, G., Gerson, I., Massey, N.: Text-to-Speech Conversion with Neural Networks: A Recurrent Approach. Proc. Eurospeech '97, pp. 561-564 (1997).
6. Bullinaria, J.: Modeling Reading, Spelling, and Past Tense Learning with Artificial Neural Networks. Brain and Language, Vol. 59, pp. 236-266 (1997).
7. Parisi, R., Di Claudio, E., Lucarelli, G.: Car Plate Recognition by Neural Networks and Image Processing. IEEE International Symposium on Circuits and Systems, Vol. 3, pp. 195-198 (1998).
8. Narendra, K.: Identification and Control of Dynamical Systems Using Neural Networks. IEEE Transactions on Neural Networks, Vol. 1, pp. 4-27 (1990).
9. Fierro, R., Lewis, F.: Control of a Nonholonomic Mobile Robot Using Neural Networks. IEEE Transactions on Neural Networks, Vol. 9, pp. 589-600 (1998).
10. Uchitel, O.: El Lenguaje de las Neuronas. Universidad De Buenos Aires (2006).
11. Haykin, Simon: Neural Networks: A Comprehensive Foundation. McMaster University, Canadá (1999).
12. ITU - R: International Morse Code Recommendation. International Telecommunication Union (2011).
13. Costa, P.: Avances y Avalanchas del Siglo XIX: Del Telégrafo Eléctrico al Teléfono. EUITT, Madrid (2011).
14. Velazquez, A., Sossa, H., Levachkine, S.: Reconocimiento Eficiente de Caracteres Alfanumericos Provenientes de Mapas Raster por Medio de Clasificadores Neuronales. Computación y sistemas, Vol. 6, Num 1, pp 38-50 (2002).
15. Melacci, S., Sarti, L., Maggini, M., Bianchini, M.: A Neural Network Approach to Similarity Learning. Artificial Neural Networks in Pattern Recognition, Third IAPR Workshop, ANNPR 2008, LNAI 5064, pp. 133-136 (2008).
16. Perez, L., Huertas, J., Lavado, A.: Redes Neuronales Aplicadas a la Demodulación en Canal Gausiano. Universidad Carlos III, Madrid (2008). Url: http://www.it.uc3m.es/jvillena/irc/practicas/07-08/DemodulacionConRedesNeuronales.pdf
17. Hsin, F., Lin, Y., Pao, H.: Neural Nets for Radio Morse Code Recognizing. Proc. SPIE 1965, Applications of Artificial Neural Networks IV, pp. 334 (1993).
18. Abdelbaki, H., El-Khamy, S.: Random Neural Network Decoder for Error Correcting Codes. Proceedings of the International Joint Conference on Neural Networks, Washington, DC, July 1999, Vol. 5, pp. 3241-3245 (2009).
19. Hassan, M., Mohammed, A.: Conversion of English Characters into Braille Using Neural Network. IJCCCE, Vol.11, Num. 2 (2011).
20. Roa, J., Gutierrez, M., Stegmayer, G.: Framework para la Enseñanza de Agentes en IA. Revista Iberoamericana de Informática Educativa, Num. 7/8, pp. 43-56 (2008).