UNIVERSIDAD TECNOLÓGICA NACIONAL

Facultad Regional Santa Fe

Doctorado en Ingeniería

Mención en Ingeniería en Sistemas de Información

Tesis Doctoral

"Desarrollo de nuevos modelos y algoritmos basados en redes neuronales para tareas de minería de datos"

Ing. Mariano Rubiolo

Director Dra. Georgina Stegmayer

Santa Fe de la Vera Cruz, Argentina. Marzo de 2014

Rubiolo, Mariano

Desarrollo de nuevos modelos y algoritmos basados en redes neuronales para tareas de minera de datos. - 1a ed. - Santa Fe de la Vera Cruz: el autor, 2014.

188 p. ; 21x29 cm.

ISBN 978-987-33-4640-8

1. Ciencias. 2. Tesis de Doctorado. I. Título
CDD 507

Fecha de catalogación: 17/03/2014

UNIVERSIDAD TECNOLÓGICA NACIONAL

Facultad Regional Santa Fe

Comisión de Posgrado

Se presenta esta Tesis en cumplimiento de los requisitos exigidos por la Universidad Tecnológica Nacional para la obtención del grado académico de Doctor en Ingeniería, mención Sistemas de Información

"Desarrollo de nuevos modelos y algoritmos basados en redes neuronales para tareas de minería de datos"

por

Ing. Mariano Rubiolo

Directora Dra. Georgina Stegmayer

Jurados de Tesis

Dra. Nélida Beatriz Brignole

Dr. Hugo Leonardo Ruffiner

Dr. Leandro Daniel Vignolo

Santa Fe de la Vera Cruz, Argentina. Marzo de 2014

A Dios.

A mi Familia.

Índice General

Índice de Tablas

Índice de Figuras

Prólogo

En muchos dominios de aplicación se almacenan grandes cantidades de datos, que necesariamente deben ser analizados en determinado momento para poder extraer información implícita, previamente desconocida y potencialmente útil, a través de un proceso que se conoce como minería de datos. La minería de datos ha proporcionado con éxito soluciones para encontrar información oculta en los datos en áreas tales como la bioinformática, farmacéutica, financiera, deportes y entretenimientos, entre otros.

Una de las tareas más conocidas en minería de datos es la clasificación que implica etiquetar (clasificar) una serie de datos utilizando una entre varias categorías (clases). En particular, para tareas de clasificación, los modelos neuronales de tipo Perceptrón Multicapa (MLP, del inglés Multilayer Perceptron) se han utilizado tradicionalmente como clasificadores dado que son modelos que han demostrado ser potentes y robustos para esa tarea.

En este contexto, en esta Tesis Doctoral se proponen nuevos modelos y algoritmos basados en redes neuronales para tareas de clasificación dentro de la minería de datos, los cuales fueron abordados desde diversos campos de aplicación. Al inicio de la tesis se comenzó a trabajar en paralelo en el diseño de clasificadores neuronales y en un método para su compresión. Luego se trabajó más fuertemente durante el desarrollo de la tesis principalmente en el tema de compresión de modelos neuronales, aplicados a diversos problemas artificiales y reales. Más recientemente se comenzó a abordar el desarrollo de modelos neuronales para minería de relaciones entre datos variantes en el tiempo, con aplicación particular a la bioinformática.

Con respecto a la tarea de clasificación y compresión de clasificadores, es sabido que los arreglos y ensambles de clasificadores han probado ser mucho más eficientes y eficaces para manejar problemas de clasificación complejos y altamente dimensionales que los modelos simples. Sin embargo, los modelos neuronales para clasificación de tipo perceptrón multicapa fácilmente se convierten en mo-

delos grandes y complejos dado que, dependiendo del problema, es posible que necesiten un gran número de parámetros y/o la introducción de más información al modelo para obtener un rendimiento aceptable. En esta tesis se presenta un nuevo modelo denominado *Volterra-NN* que tiene por objetivo comprimir un clasificador neuronal en un menor número de parámetros, manteniendo al mismo tiempo las altas tasas de desempeño del clasificador original. Luego, se propone una extensión el modelo antes mencionado para lograr comprimir modelos clasificadores de mayor complejidad y de más variada aplicabilidad, tal como un arreglo de clasificadores neuronales. Así surge el modelo aV-NN, que logra con el mismo éxito altas tasas de reconocimiento, habiendo sido probado tanto en ejemplos artificiales de diversa complejidad de la literatura, como también en una aplicación real en el área de la biometría, como es el reconocimiento de rostros. La propuesta se compara con dos algoritmos clásicos relacionados a la compresión de modelos neuronales. También se propone una medida de selección de la mejor solución que permite al usuario determinar de entre todos los posibles modelos, el que mejor se ajusta a sus requerimientos.

Como se mencionó anteriormente, al principio de la tesis se trabajó en el diseño de un clasificador neuronal aplicado al problema de la correspondencia entre ontologías, en base a un modelo perceptrón multicapa. La información de entrada al clasificador es codificada mediante el uso de un corpus reconocido. Se aplica el enfoque propuesto en un dominio real de Investigación y Desarrollo, con anotaciones semánticas de ontologías utilizadas y reconocidas en la literatura. Al final, en base a los resultados, se compara el enfoque propuesto con un algoritmo clásico de correspondencia ontológica.

Finalmente, se presenta un enfoque basado en la utilización de redes neuronales para modelar las relaciones existentes entre series temporales en el área de la Bioinformática. Se utiliza un conjunto de redes perceptrón multicapa para analizar todas las posibles combinaciones de relaciones gen a gen entre los genes presentes en el conjunto de datos y luego se aplican reglas de minería de relaciones entre los genes. Este método es probado sobre un conjunto de datos artificiales surgidos a partir de un simulador, y sobre un conjunto de datos biológicos perteneciente a los perfiles de expresión de genes del tomate. Esta propuesta también se compara con otros trabajos relacionados y se reportan los resultados.

Los resultados parciales del trabajo realizado en la tesis han sido divulgados a través de las siguientes publicaciones:

- Rubiolo, M.; Stegmayer, G. y Milone, D. H. (2013). «Compressing arrays of classiffiers using Volterra-Neural Network: application to face recognition». *Neural Computing & Applications*, 23(6), pp. 1687-1701. Factor de Impacto 2012: 1.168.

- Rubiolo, M.; Caliusco, M.L.; Stegmayer, G.; Coronel, M. y Fabrizi, M. Gareli (2012). «Knowledge discovery through ontology matching: An approach based on an Artificial Neural Network model». *Information Sciences*, 194, pp. 107-119. Factor de Impacto 2012: 3.643.

- Rubiolo, M.; Stegmayer, G. y Milone, D. (2013). «A novel approach for gene regulatory network mining from temporal series data using neural networks». *IEEE Transactions on Computational Biology and Bioinformatics*. En evaluación.

- Rubiolo, M. (2012). «Extended evaluation of the Volterra-Neural Network for model compression». En: *13th Argentine Symposium on Artificial Intelligence (ASAI 2012) on 41th Argentine Conference on Informatics*, pp. 1-12.

- Rubiolo, M.; Stegmayer, G. y Milone, D. (2010). «Compressing a Neural Network Classiffier using a Volterra-Neural Network model». En: *IEEE International Joint Conference on Neural Networks (IJCNN)*, pp. 1-7.

- Rubiolo, M.; Caliusco, M.L.; Stegmayer, G.; Gareli, M. y Coronel, M. (2009). «Knowledge Source Discovery: An experience using Ontologies, WordNet and Artificial Neural Networks». En: *13th International Conference on Knowledge-Based and Intelligent Information & Engineering Systems (KES)*, pp. 11-17.

Resumen

En esta Tesis Doctoral se propone estudiar y desarrollar modelos neuronales para diversas tareas de clasificación dentro de la minería de datos. En primer lugar, se propone el modelo *Volterra-NN* para comprimir un clasificador neuronal, manteniendo altas tasas de desempeño. Se extiende la aplicación a clasificadores de mayor complejidad con el modelo aV-NN, logrando exitosamente la compresión de un arreglo de clasificadores, tanto en ejemplos de diversa complejidad surgidos de la literatura, como también en una aplicación real de reconocimiento de rostros. En segundo término, se propone un clasificador neuronal para resolver la correspondencia entre ontologías, aplicándose en un dominio real de I+D con anotaciones semánticas de ontologías utilizadas en la literatura. Por último, se presenta un enfoque basado en la utilización de redes neuronales para modelar las relaciones existentes, pero desconocidas de antemano, entre series temporales en el área de la Bioinformática, probándose sobre un conjunto de datos artificiales simulados, y de datos biológicos reales.

Agradecimientos

Quiero expresar mi gratitud, en primer lugar, a Dios, por regalarme la posibilidad de lograr esta importante meta en mi vida.

Estoy agradecido profundamente con Marita, Jerónimo y Augusto, mi familia, a quienes también pertenece el tiempo dedicado a esta tesis. Sin su amor y apoyo incondicional hubiera sido imposible realizarla. También quiero agradecer a mis padres y hermanos, los que siempre han alentado y acompañado cada uno de mis logros, no sólo en lo profesional, sino en la vida misma.

Agradezco a mi directora, Dra. Georgina Stegmayer, quien no sólo creyó en mi capacidad para recorrer con éxito este camino, sino que no ha escatimado nunca en ofrecerme sus valiosos conocimientos, su experiencia profesional, su apoyo constante, sus consejos oportunos, su paciencia y cariño, tanto para el desarrollo de la tesis como para lo compartido cotidianamente.

Mi agradecimiento a todos y cada uno de los integrantes del Centro de Investigación y Desarrollo en Ingeniería en Sistemas de Información CIDISI, perteneciente a la UTN-FRSF, por brindarme el espacio y soporte necesarios para poder desarrollar mis investigaciones. En particular, a la Dra. María Laura Caliusco, por compartir su tiempo y sus conocimientos, que fueron fuente de inspiración de trabajos en conjunto, cuyos resultados forman parte de esta tesis.

También quiero agradecer al Centro de Investigación en Señales, Sistemas e Inteligencia Computacional SINC(i), de la FICH-UNL, por brindarme un espacio de colaboración muy importante. Especialmente al Dr. Diego Milone, quien ha colaborado conmigo en trabajos que se documentan en esta tesis, facilitándome sus conocimientos, compartiendo su valiosa experiencia, y nutriéndome de sus valores profesionales y personales.

No quiero dejar de agradecer a las instituciones que me han otorgado el soporte económico para el desarrollo de mi Doctorado. A la Agencia Nacional de Promoción Científica y Tecnológica (ANPCyT) y a la Universidad Tecnológica Nacional por el otorgamiento de su beca para la Formación de Posgrado destina-

da a Docentes de la UTN. Al Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET) por otorgarme su Beca Interna de Posgrado Tipo II.

A mis "compañeros de ruta" Ivanna, Jorge, y Edgar, con quienes he compartido cada sueño en lo profesional, los momentos difíciles, y las alegrías y satisfacciones que me ha otorgado esta formación de posgrado. A mis compañeros del grupo de Bioinformática: Milton, Guillermo, David y Matías, quienes han colaborado conmigo estos últimos años, siendo testigos de los avances y "retrocesos" de mis investigaciones.

Por último, muchas gracias a mis Amigos, quienes se alegran con cada paso que doy en la vida, no sólo en lo profesional, sino también en mi constante desarrollo como Persona.

Mariano Rubiolo
Santa Fe de la Vera Cruz, Argentina
Noviembre 2013

CAPÍTULO 1

Introducción

El presente capítulo describe la motivación que dió lugar a la tesis (Sección 1.1), y los objetivos planteados en el trabajo de investigación (Sección 1.2). Además, se mencionan las principales contribuciones alcanzadas (Sección 1.3), y se presenta la organización general de la tesis (Sección 1.4).

1.1. Motivación

La minería de datos se refiere al análisis de una gran cantidad de datos almacenados en computadoras (Olson y Delen, 2008). Dicho proceso implica la extracción de información implícita, previamente desconocida y potencialmente útil de esa gran cantidad de datos (Witten y Frank, 2005).

Existen diferentes maneras de clasificar las tareas de minería de datos. En general, se adopta la categorización que captura los diferentes tipos de procesos involucrados, como son, pre-procesamiento de los datos, modelado de los datos y descripción del conocimiento (Xu y Wunsch, 2009). Tal como se puede observar en la Figura 1.1, el pre-procesamiento de los datos por lo general incluye la eliminación de ruido, selección de características, integración de los datos y tratamiento de datos faltantes. En cuanto al modelado de datos, éste incluye a la clasificación, el agrupamiento, y la predicción de clases. La descripción del conocimiento involucra técnicas que permitan explicitar e interpretar las relaciones descubiertas en los datos.

Es posible definir a una clase como un conjunto de datos con cierto grado de similitud o relación. A todas las muestras de una clase se les asigna la misma etiqueta para distinguirlas de las muestras en otras clases. Por su parte, un grupo (o cluster) es una colección de objetos que son similares a nivel local. Estas agrupaciones suelen ser generadas con el fin de clasificar objetos en ca-

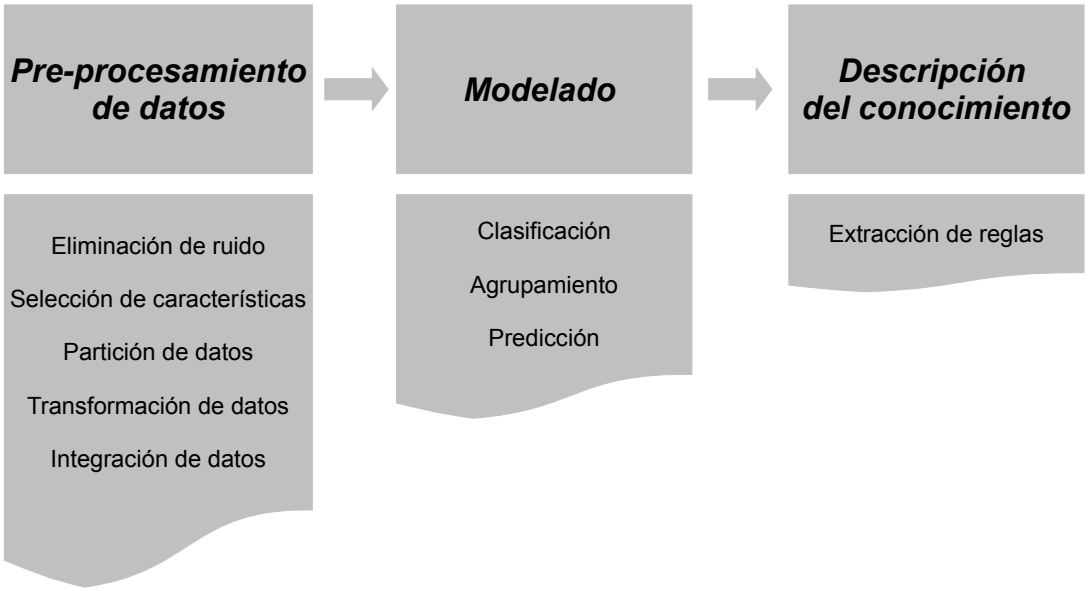

Figura 1.1: Proceso de Minería de Datos.

tegorías relativamente mayores y más significativas. Dado un conjunto de datos con etiquetas de clase, se construyen clasificadores como predictores para futuros objetos desconocidos. De este modo, un modelo de clasificación está formado primeramente sobre la base de datos disponibles y las tendencias futuras se prevén usando el modelo aprendido (Duda y Hart, 2003).

Desde el punto de vista del aprendizaje de máquina y el aprendizaje estadístico, la clasificación de patrones es un problema difícil de atacar computacionalmente (Olson y Delen, 2008). Para un problema tipo, por lo general hay más de un centenar de muestras en un conjunto de datos. Por lo tanto, ese problema se convierte en un problema de reconocimiento de patrones con un número de patrones que, por lo general, es de alta dimensión (Tasoulis y otros, 2008).

La minería de datos ha proporcionado con éxito soluciones para encontrar información oculta en los datos en áreas tales como la bioinformática, farmacéutica, financiera, deportes y entretenimientos, entre otros. Es uno de los campos que ha crecido más velozmente en los ultimos años. Muchos problemas importantes en la ciencia y la industria han sido abordados mediante métodos de minería de datos, tales como redes neuronales, lógica difusa, árboles de decisión, algoritmos genéticos y métodos estadísticos (Wang y Fu, 2005).

En particular, para tareas de clasificación, los modelos neuronales de tipo Perceptrón Multicapa (MLP, del inglés Multilayer-Perceptron) se han utilizado

tradicionalmente como clasificadores dado que son modelos que han demostrado ser potentes y robustos (Haykin, 2007; Widrow y Lehr, 2002). Por otro lado, los arreglos y ensambles de clasificadores han probado ser mucho más eficientes y eficaces para manejar problemas de clasificación complejos y altamente dimensionales, alcanzando altas tasas de clasificación, en comparación a los modelos clásicos (Shimshoni y Intrator, 1996). Un arreglo de clasificadores está compuesto por varios clasificadores más simples, por ejemplo, MLPs especializados cada uno en una clase. Un ensamble consiste en varios clasificadores base que aprenden los límites de decisión sobre las clases de los patrones de entrenamiento, y sus decisiones cuando se les presenta un patrón de prueba se fusionan para llegar a la clasificación final (Rahman y Verma, 2011). Sin embargo, los modelos neuronales fácilmente se convierten en modelos grandes dado que, dependiendo del problema, es posible que necesiten un gran número de parámetros (diversas funciones de activación en las neuronas, pesos sinápticos y umbrales) para obtener un rendimiento aceptable. Del mismo modo, la introducción de más información al modelo, es decir, más variables de entrada o más capas y neuronas, con el fin de mejorar los datos de entrenamiento y mejorar su capacidad de clasificación y generalización, puede causar un incremento de la complejidad del modelo (Buciluǎ y otros, 2006a).

Por todo lo expuesto, en el marco de esta Tesis Doctoral se propuso estudiar y trabajar sobre modelos neuronales utilizados para diversas tareas de clasificación dentro de la minería de datos. Se desarrolló, en primer lugar, un nuevo modelo denominado *Volterra-NN*, que permite comprimir un clasificador (un modelo neuronal MLP) en un menor número de parámetros, manteniendo al mismo tiempo las altas tasas de desempeño del clasificador. Utilizando estos parámetros, se construyen redes de Volterra de diferente orden, las cuales permiten comprimir significativamente el número de parámetros necesarios para mantener altas tasas de clasificación. En segundo término, se propuso un clasificador neuronal para resolver el problema de la correspondencia entre ontologías. Por último, se presentó un enfoque basado en redes neuronales para modelar las relaciones existentes entre series temporales en el área de la Bioinformática.

1.2. Objetivos

El objetivo general de esta tesis es el desarrollo de nuevos modelos, algoritmos y herramientas computacionales basados en Redes Neuronales Artificiales, aplicables al análisis, clasificación y reconocimiento de patrones de comportamiento en registros de datos de diverso tipo, cuyo tratamiento con los métodos actuales sea inadecuado o insuficiente.

A continuación se detallan los objetivos específicos propuestos al comienzo de la ejecución de la presente investigación:

- Desarrollo de nuevos modelos y algoritmos para clasificación, que permitan mantener altas tasas de clasificación empleando menos parámetros que los modelos tradicionales.
- Validación de los resultados obtenidos con los modelos propuestos en relación a los obtenidos con modelos tradicionales a través de ejemplos simples.
- Extensión de la aplicación de los modelos propuestos a problemas de clasificación complejos en los cuales los clasificadores sean, por ejemplo, arreglos de redes neuronales.
- Validación de las relaciones entre datos que pudieran encontrarse mediante la aplicación de las técnicas propuestas a casos de estudio artificiales y reales, que involucren un gran número de clases.
- Implementación e integración de los métodos propuestos en una herramienta computacional.

1.3. Principales contribuciones

En el marco de esta Tesis Doctoral, se trabajó principalmente sobre modelos neuronales en diversos campos de aplicación y en distintos momentos de tiempo. Al inicio de la tesis, se comenzó a desarrollar en paralelo un modelo para compresión de clasificadores neuronales y un clasificador neuronal para resolver la correspondencia entre ontologías. En la última etapa de esta tesis, se inició el abordaje del uso de redes neuronales para minería de relaciones entre series temporales en problemas de bioinformática.

En cuanto al nuevo modelo denominado *Volterra-NN*, que permite comprimir un clasificador en un menor número de parámetros manteniendo al mismo tiempo las altas tasas de desempeño del clasificador, han sido probados y validados en problemas simples y típicos del área de machine learning [1] , en particular sobre la base de datos Iris[2] . Estos primeros resultados han sido comunicados en un congreso internacional con referato (Rubiolo y otros, 2010). La extensión de estos modelos a problemas más complejos que involucran más clases y más variables, y su aplicación a la compresión de arreglos de clasificadores, fue realizado con un arreglo de Volterra-NN (aV-NN), el cual permite escalar en la complejidad del problema a resolver cuando éste involucra varias clases, logrando también altas tasas de compresión sin que esto reduzca las tasas de clasificación. Este nuevo modelo ha sido probado y validado en problemas clásicos de reconocimiento de patrones estudiados frecuentemente en la literatura, tales como Letter Recognition[3] (Letters), Pen-Based Recognition of Handwritten Digits[4] (Pendigits), y también ha sido aplicado en un problema biométrico como el reconocimiento de rostros, utilizando las bases de datos AT&T Laboratories Cambridge ORL Database of Faces[5] , y Facial Recognition Technology (FERET)[6] (Phillips y otros, 2000). Como resultados, han surgido una comunicación en un congreso nacional con referato (Rubiolo, 2012), y una publicación en una revista indexada con Factor de Impacto 2012 de 1.168 (Rubiolo y otros, 2013b).

Paralelamente, se ha propuesto un modelo clasificador neuronal para resolver el problema de la correspondencia ontológica, basado en un modelo Perceptrón Multicapa. Se realizó una asociación entre este modelo y el de correspondencia ontológica. Además, se propuso la codificación de las entradas al modelo neuronal mediante el uso del corpus Wordnet[7] , lo cual lo independiza del idioma. Se aplicó el modelo propuesto a un problema real en el dominio de I+D con anotaciones semánticas de ontologías utilizadas en la literatura[8] [9] [10] , obteniéndose

[1] UCI Machine Learning Repository: http://archive.ics.uci.edu/ml/
[2] http://archive.ics.uci.edu/ml/datasets/Iris
[3] http://archive.ics.uci.edu/ml/datasets/Letter+Recognition
[4] http://archive.ics.uci.edu/ml/datasets/Pen-Based+Recognition+of+Handwritten+Digits
[5] http://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html
[6] http://www.itl.nist.gov/iad/humanid/feret
[7] http://wordnet.princeton.edu/
[8] http://protege.cim3.net/file/pub/ontologies/ka/ka.owl
[9] http://ontoware.org/projects/swrc/
[10] http://libertad.univalle.edu/jsequeda/ontology/olid.owl

resultados satisfactorios no sólo en cuanto a la capacidad del modelo neuronal de aprender correctamente la información que contiene cada dominio y ontología anotada, sino también en comparación con un algoritmo de búsqueda de correspondencias entre ontologías reconocido en la literatura, llamado H-Match. Estos resultados han sido comunicados a un congreso internacional con referato (Rubiolo y otros, 2009) y publicados también en una revista indexada con Factor de Impacto 2012 de 3.643 (Rubiolo y otros, 2012).

Más recientemente se ha trabajado en un enfoque novedoso para solucionar un problema de gran interés actualmente en Bioinformática, como es el reconstruir una red de regulación de genes (GRN) a partir de los datos de expresión de varios genes candidatos. Este enfoque requirió del modelado de la interacción entre cada par de genes utilizando una RNA. Específicamente, se utilizó un conjunto de redes perceptrón multicapa para analizar todas las posibles combinaciones de relaciones gen a gen entre los genes presentes en el conjunto de datos. La capacidad de modelar efectivamente cada relación fue medida de acuerdo al error de generalización de cada modelo neuronal, obteniéndose así un ranking de los modelos con menor error cometido en el entrenamiento. Sobre todos estos posibles resultados se aplicaron un conjunto de reglas de minería de relaciones para descubrir la GRN subyacente. El método propuesto fue probado realizando varios experimentos sobre un conjunto de datos artificiales surgidos a partir de un simulador (Smith y otros, 2002), que representan a una GRN previamente conocida. Además, se validó la propuesta con un conjunto de datos biológicos reales perteneciente a los perfiles de expresión de genes del tomate involucrados en la producción de Vitamina E (Quadrana y otros, 2013). Los resultados, en ambos casos, mediante el cálculo de las medidas de desempeño de sensitividad y precisión, demostraron que la propuesta es capaz de descubrir efectivamente todas las relaciones gen a gen existentes y reconstruir la GRN subyacente en los datos. Esto ha sido enviado a una revista internacional indexada con Factor de Impacto 2012 de 1.616 (Rubiolo y otros, 2013a) y al momento de escritura de la presente tesis se encuentra en proceso de revisión.

1.4. Organización de la tesis

La organización de la presente tesis es la siguiente.

El capítulo 2 tiene por objetivo presentar el marco teórico de los conceptos utilizados a lo largo de la tesis. En primer lugar se describen los principios de los modelos de Redes Neuronales Artificiales. Luego, se presenta el modelo de serie de Volterra y se explica cómo se puede obtener los kernels de la serie desde un modelo neuronal entrenado. Por último, se presentan las medidas utilizadas para evaluar los algoritmos, modelos y metodologías propuestos.

El capítulo 3 explica el nuevo modelo propuesto basado en redes neuronales para comprimir clasificadores, el modelo de Redes Neuronales de Volterra (Volterra-NN). Por otra parte se propone una nueva medida de desempeño para determinar la solución de mejor compromiso entre la compresión del modelo y la precisión en la clasificación. Se demuestra la aplicabilidad, viabilidad y utilidad de los modelos y algoritmos propuestos para compresión de modelos, a través del análisis de los resultados obtenidos en su aplicación a diversos problemas, de amplia variedad y complejidad.

El capítulo 4 presenta un clasificador de aprendizaje maquinal basado en redes neuronales, utilizado para definir la operación de correspondencia entre ontologías. Se explica cómo el método propuesto tiene en cuenta tanto la información a nivel de esquema como a nivel de instancia, a partir de las ontologías y las anotaciones semánticas; y es probado en un caso de estudio real. Además, se comparan los resultados obtenidos con los de un algoritmo de correspondencia ontológica reconocido en la literatura.

En el capítulo 5 se propone un nuevo enfoque para minería de relaciones entre perfiles de expresión de genes utilizando un conjunto de redes neuronales artificiales. Se explica en detalle como se entrena un modelo neuronal utilizando los datos temporales de los perfiles de expresión para el modelado de las relaciones gen a gen, con el objetivo de descubrir la red de regulación de genes subyacente en el conjunto de datos. Para su validación, se utilizó un conjunto de datos artificiales con interacciones conocidas previamente, con el objeto de confirmar la precisión y la sensitividad de la propuesta. Por último, se aplicó el enfoque a un conjunto de datos biológicos reales.

El capítulo 6 resalta las principales contribuciones de la tesis y describe los aspectos de la misma que constituyen el punto de partida para el desarrollo de los trabajos futuros.

CAPÍTULO 2

Marco Teórico

2.1. Introducción

El presente capítulo describe el marco teórico de la investigación realizada, exponiendo conceptos y detalles de modelos y algoritmos necesarios para entender los aportes realizados en esta tesis. Se describen los principios de los modelos de Redes Neuronales Artificiales en la Sección 2.2. En la Sección 2.3 se describe el modelo de serie de Volterra y cómo se pueden obtener sus parámetros a partir de un modelo neuronal entrenado. Por último, se presentan las medidas de desempeño utilizadas para evaluar los modelos y algoritmos propuestos en la tesis en la Sección 2.4.

2.2. Redes Neuronales Artificiales

Las Redes Neuronales Artificiales (RNAs) son un paradigma computacional en el cual una gran cantidad de unidades de cómputo simple (las neuronas) interconectadas en red, realizan tareas de procesamiento de datos. Una RNA puede definirse formalmente como un procesador paralelo y distribuido que tiene la capacidad de almacenar conocimiento basado en la experiencia. Además, se parece al cerebro humano en cuanto a que el conocimiento es adquirido a través de un proceso de aprendizaje y las conexiones entre las neuronas son usadas para almacenar ese conocimiento (Haykin, 2007). Las principales características de las RNAs pueden resumirse en los siguientes items:

- Habilidad de aprender mediante ejemplos o experiencias.
- Son adaptables, debido a la capacidad de autoajuste de las neuronas, y dinámicas ya que son capaces de adaptarse a nuevos datos.

- Habilidad para generalizar, es decir asociar entradas (inputs) similares con salidas (outputs) similares.
- Robustez ante fallos, dado que pueden seguir realizando su función, con cierta degradación, aunque se destruya parte de la red ya que la información se encuentra distribuida en las conexiones entre neuronas.
- Capacidad para reconocer patrones con ruido, distorsionados o incompletos.
- Pragmatismo, es decir, se prioriza encontrar una buena solución de forma relativamente rápida a encontrar la mejor solución pero en forma más lenta.
- Capacidad de modelar problemas que involucren una o múltiples variables.

En muchas áreas de aplicación, el conocimiento que se necesita para resolver un problema puede estar incompleto debido a que la fuente de ese conocimiento es desconocida, o el entorno puede ser cambiante y difícil de ser descripto en tiempo de diseño. Existen problemas ingenieriles en los que para encontrar una solución óptima se requiere una cantidad de recursos (o tiempo) prácticamente imposibles de conseguir, siendo preferible la obtención de una solución aceptable. Las RNAs pueden brindar buenas soluciones para tales clases de problemas. Las RNAs parten de un conjunto de datos de entrada y el objetivo del entrenamiento es conseguir que la red aprenda esos datos de entrada. El diseño de la red implica la selección del modelo o tipo de red, las variables a incorporar y el pre-procesamiento de la información que formará parte del conjunto de entrenamiento. Las RNA implementan algoritmos que intentan alcanzar cierto desempeño aprendiendo de la experiencia, formulando generalizaciones desde situaciones similares y penalizando estados en los que se han alcanzado resultados pobres (Chaturvedi, 2008).

Dado que las RNAs pueden aprender y resolver problemas complejos, han sido aplicadas en campos tan diversos como el reconocimiento de patrones, clasificación, el procesamiento del habla, el control de sistemas, aplicaciones médicas, bioinformática, entre muchos otros (Bishop, 1995; Haykin, 2008). Además, han sido aplicadas a gran cantidad de problemas, no sólo en la academia sino también en la industria (Meireles y otros, 2003).

A continuación se describirán sus elementos básicos, tipos de modelos, forma de funcionamiento y reglas de aprendizaje.

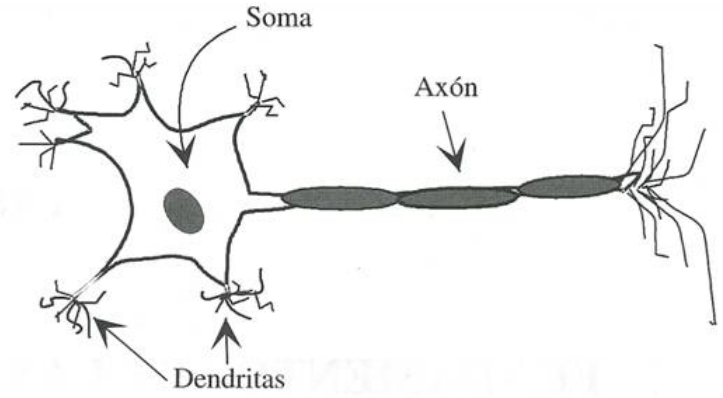

Figura 2.1: Estructura de una neurona biológica típica (Brío y Molina, 2007).

2.2.1. Elementos básicos.

Las RNAs se encuentran inspiradas en el modelo biológico de las neuronas del cerebro. El cerebro humano está compuesto por más de cien mil millones de neuronas interconectadas entre sí formando circuitos o redes que desarrollan funciones específicas. Como se muestra en la Figura 2.1, una neurona biológica típica recoge señales procedentes de otras neuronas a través de las dendritas. La neurona emite impulsos de actividad eléctrica a lo largo de una fibra larga y delgada denominada axón, que se esconde en millares de ramificaciones axonales. Las extremidades de estas ramificaciones llegan hasta las dendritas de otras neuronas y establecen unas conexiones llamadas sinapsis, en las cuales se produce una transformación del impulso eléctrico en un mensaje neuroquímico, mediante la liberación de unas sustancias llamadas neurotransmisores. Los neurotransmisores liberados en las sinapsis al final de las ramificaciones axonales pueden tener un efecto excitatorio o inhibitorio sobre la neurona receptora.

Las señales recibidas por una neurona se combinan, y en función de la estimulación total recibida, la neurona toma un cierto nivel de activación, que se traduce en la generación de breves impulsos nerviosos con una determinada frecuencia o tasa de disparo, y su propagación a lo largo del axón hacia las neuronas con las cuales se conecta. De esta manera la información se transmite de unas neuronas a otras y va siendo procesada a través de las conexiones sinápticas y las propias neuronas. El aprendizaje de las redes neuronales se produce mediante la variación de la efectividad de las sinapsis. De esta manera, cambia la influencia que unas neuronas ejercen sobre otras. De aquí se deduce que la arquitectura, el

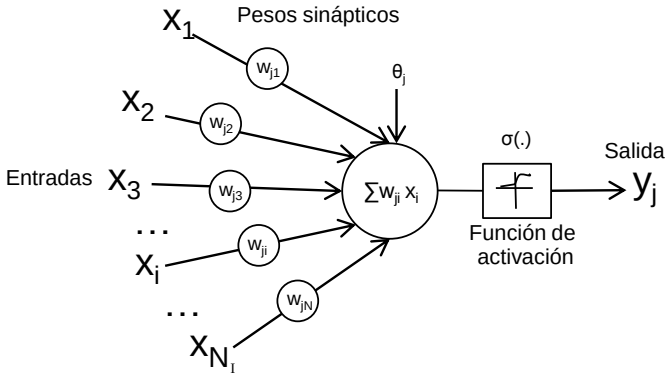

Figura 2.2: Modelo de Neurona artificial (Haykin, 2008).

tipo y la efectividad de las conexiones en un momento dado, representan en cierto modo la memoria o estado de conocimiento de la red (Brío y Molina, 2007).

Una RNA está compuesta por un conjunto de neuronas artificiales interconectadas entre sí, emulando conexiones sinápticas y liberación de neurotransmisores con efectos excitatorios o inhibidores. Cada neurona artificial se representa como una unidad compuesta de conexiones de entrada, un núcleo central de proceso y una salida (Rumelhart y otros, 1986). Como se ve en la Figura 2.2, una neurona artificial consta de una serie de entradas x_i (i = [1,...,N_I]), que equivalen a las dendritas de donde reciben la estimulación; ponderadas por los w_{ji} que pesan la señal que pasa hacia la neurona j proviniente de la entrada i, que representan cómo los impulsos entrantes son evaluados; combinados con el *umbral* θ_j, lo que nos dará el nivel de potencial de la neurona. El resultado de esta sumatoria es evaluada en la *función de activación* $\sigma(x)$ que da lugar a la salida de la unidad de proceso y_j. En términos matemáticos y en base a los elementos recién descriptos, una neurona artificial j se puede describir como:

$$y_j = \sigma\left(\sum_{i=1}^{N_I} w_{ji}x_i \pm \theta_j\right). \tag{2.1}$$

Las entradas pueden ser valores enteros, reales o binarios. Estos valores equivalen a las señales que enviarían otras neuronas a la neurona artificial y_j a través de las dendritas. Los pesos que hay en las sinapsis equivaldrían en la neurona biológica a los neurotransmisores liberados en la transformación del impulso eléctrico. De forma que la unión de estos valores (x_i y w_{ji}) equivalen a las señales químicas inhibitorias y excitadoras que se dan en las sinapsis, que inducen a la

neurona y_j a cambiar su comportamiento, y que suele referirse a ellos como el peso sináptico w_{ji}.

El peso sináptico w_{ji} define la fuerza de una conexión sináptica entre dos neuronas, la neurona postsináptica j y la neurona presináptica i. Los pesos sinápticos pueden tomar valores positivos, negativos o cero. En el caso de una entrada positiva (procedente de una entrada o simplemente la salida de otra neurona), un peso positivo actúa como excitador, mientras que un peso negativo actúa como inhibidor. En caso de que el peso sea cero, no existe comunicación entre el par de neuronas i y j. Los pesos pueden ser modificados en respuesta al entrenamiento que se le aplique a la RNA, dependiendo de la topología específica de la red o de sus reglas de aprendizaje. Con frecuencia se añade al conjunto de pesos de la neurona un parámetro adicional θ_j, denominado umbral (threshold o bias), que suma o resta del potencial postsináptico, lo que representa añadir un grado de libertad adicional a la neurona. En muchos casos este parámetro representará el umbral de disparo de la neurona, es decir, el nivel mínimo que debe alcanzar el potencial postsináptico para que la neurona se dispare o active (Brío y Molina, 2007).

Como ya se dijo, la suma ponderada de las entradas y los pesos sinápticos equivale a la combinación de las señales excitatorias e inhibitorias de las neuronas biológicas. Esta suma llega a la función de activación $\sigma(x)$ que se encarga de calcular el nivel o estado de activación de la neurona en función de la entrada total. Para ello, se pueden utilizar diferentes tipos de funciones, desde simples funciones de umbral a funciones no lineales como la tangente hiperbólica o sigmoidea. La función de activación es normalmente monótona creciente y continua, como se observa habitualmente en las neuronas biológicas. Algunas variantes comunes para la función de activación σ(x) puede ser alguna de las siguientes:

- Escalón

$$\sigma(x) = \begin{cases} 1 & \text{if } x \geq 0; \\ 0 & \text{if } x < 0 \end{cases} \tag{2.2}$$

- Lineal

$$\sigma(x) = x \tag{2.3}$$

- Sigmoidea

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.4}$$

- Tangente hiperbólica

$$\sigma(x) = \tanh(x) = \frac{1 - e^{-x}}{1 + e^{-x}} \tag{2.5}$$

Cuando los algoritmos de aprendizaje requieren que la función de activación sea derivable (como en el algoritmo de retropropagación que se explicará más adelante) se suelen emplear la función de sigmoidea o tangente hiperbólica. La salida de una neurona artificial (al igual que las entradas) puede ser binaria (digital) o continua (analógica), dependiendo del modelo y de la aplicación. Por ejemplo, para tareas de clasificación se emplearían salidas binarias, mientras que en un problema de ajuste funcional de una aplicación multivariable continua, se emplearían salidas continuas dentro de un cierto intervalo. En el caso de que la salida sea binaria, se supone que la neurona no se dispara hasta que la activación supera cierto umbral.

Las neuronas pueden conectarse unas a otras y disponerse de tal manera que forman una estructura, topología o patrón de conexiones, lo que se denomina la arquitectura de la red neuronal. En general, y de modo análogo a lo que sucede en el cerebro, las neuronas se suelen agrupar en unidades estructurales que se denominan capas. El conjunto de una o más capas conforma la RNA. El número de capas, como así también el número de neuronas por capa, el grado de conectividad y el tipo de conexiones entre neuronas, son datos fundamentales para definir un modelo neuronal. Se pueden distinguir, en función a su topología, redes monocapa, como aquellas compuestas por una única capa de neuronas; y las redes multicapa, cuyas neuronas se organizan en varias capas. Asimismo, atendiendo al flujo de datos dentro de la red, se pueden distinguir redes unidireccionales (feedforward) y redes recurrentes (feedback). En las redes unidireccionales, la información circula en un único sentido, desde las neuronas de entrada hacia las de salida. En las redes recurrentes o realimentadas la información puede circular entre las capas en cualquier sentido, incluido el de salida-entrada.

2.2.2. Aprendizaje en las RNA.

Una característica fundamental de las RNAs es que se trata de sistemas entrenables, capaces de realizar un determinado tipo de procesamiento o cómputo, aprendiéndolo a partir de un conjunto de patrones de aprendizaje o ejemplos.

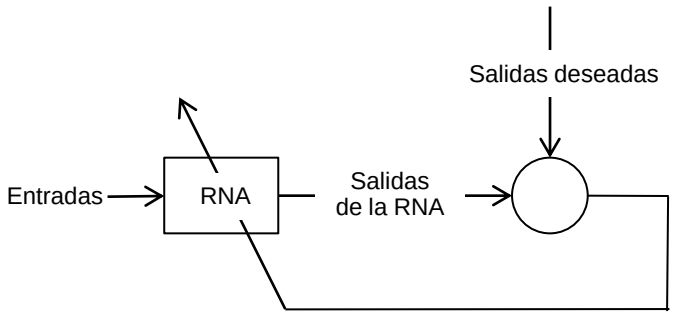

Figura 2.3: Aprendizaje supervisado.

El entrenamiento es un proceso de optimización donde se ajustan los parámetros internos del modelo neuronal (pesos y umbrales) para adaptarse a los datos de entrenamiento (Haykin, 2007). Inicialmente los pesos sinápticos se establecen como nulos o en forma aleatoria y luego, en un proceso de entrenamiento, se van ajustando los mismos hasta encontrar una respuesta aceptable para la tarea deseada. Normalmente el ajuste de los pesos sinápticos se realiza siguiendo alguna regla de aprendizaje que va midiendo el rendimiento actual de la red. Es posible identificar dos métodos de aprendizaje principales: aprendizaje supervisado y aprendizaje no supervisado.

En el *aprendizaje supervisado* se le presenta a la RNA un conjunto de patrones junto con la salida deseada u objetivo. La red iterativamente va ajustando sus pesos hasta aproximarse a la salida deseada. Los pesos se van modificando de manera proporcional al error que se produce entre la salida real de la red y la salida esperada, tal como se muestra en la Figura 2.3. Esta forma de aprendizaje también es denominado aprendizaje por corrección de errores. Como una medida del desempeño del sistema, se puede pensar en términos del error cuadrático medio (por ejemplo, el valor esperado de la suma de los errores al cuadrado) definido como una función de los parámetros libres del modelo neuronal. Esta función puede verse como una superficie de error multidimensional, con los parámetros libres como coordenadas. Cualquier configuración del modelo neuronal es representada como un punto en esa superficie. Para que el modelo mejore su desempeño a lo largo de las iteraciones de entrenamiento, ese punto debe descender sucesivamente hacia un punto mínimo de la superficie de error, el cual puede ser local o global. Un sistema supervisado de aprendizaje puede hacer esto usando información acerca del gradiente del error, o una estimación instantánea de ese gradiente a

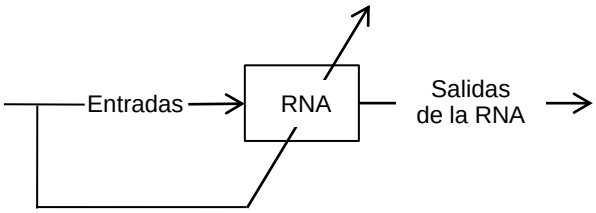

Figura 2.4: Aprendizaje no supervisado.

lo largo de las iteraciones. Dado un adecuado conjunto de datos de entrenamiento entrada-salida deseada y suficiente tiempo, un algoritmo supervisado puede usualmente realizar tareas tales como clasificación, reconocimiento de patrones y aproximación de funciones, de modo satisfactorio (Haykin, 2007).

En el *aprendizaje no supervisado* o auto-organizativo se le presenta a la RNA un conjunto de patrones pero sin la salida deseada. En este caso, no hay un conjunto de ejemplos de los cuales aprender. A través de la regla de aprendizaje definida en la arquitectura, la RNA debe reconocer en los patrones que se le presentan regularidades, extraer rasgos, estimar densidades de probabilidad o agruparlos según su similitud. Este aprendizaje se ilustra en la Figura 2.4. Para este tipo de aprendizaje, se usa una regla de aprendizaje competitiva, en la cual las neuronas compiten entre sí para activarse frente a un dato de entrada. En su forma más simple, la red opera en el modo denominado "winner-takes-all", en el cual la neurona que gana la competencia se enciende o es premiada, y las otras no.

Finalizada la etapa de entrenamiento, es importante conocer cómo responde la RNA ante datos que nunca haya visto antes, a través de la fase de *prueba*. Si la red no otorga resultados razonables para ese conjunto de datos de prueba, el entrenamiento no ha terminado. Esta fase es de suma importancia para asegurar que la red no se ha limitado a memorizar el conjunto de entrenamiento. Una vez el sistema ha sido entrenado y probado, en la mayor parte de los modelos el aprendizaje se suspende, los pesos y la estructura permanecen fijos, y la red neuronal queda dispuesta para procesar información, proporcionando una respuesta ante un patrón o vector de entrada.

2.2.3. El perceptrón simple y el modelo Perceptrón Multicapa.

El perceptrón simple es la forma más simple de RNA usada para clasificación de un tipo especial de patrones que pertenecen a dos clases linealmente separables. Básicamente consiste de una sola neurona con pesos y un umbral. El algoritmo usado para ajustar estos parámetros libres fue propuesto por Rosenblatt en 1958. De hecho, Rosenblatt probó que si los datos usados para entrenar el perceptrón pertenecen a dos clases linealmente separables, el algoritmo del perceptron converge y éste posiciona la región de decisión como un hiperplano entre las dos clases. La prueba de convergencia de este algoritmo se conoce como teorema de convergencia del perceptrón.

Uno de los tipos de RNA más comúnmente utilizados es el Perceptrón Multicapa (MLP de su nombre en inglés Multilayer Perceptron). Está formado por varias capas de neuronas del tipo perceptrón simple. Posee típicamente una capa de entrada, una o más capas ocultas y una capa de salida. El MLP es una red de tipo unidireccional que utiliza aprendizaje supervisado. Las neuronas de la capa de entrada alimentan la red distribuyendo hacia adelante las señales de entrada. Cada neurona en las capas ocultas y de salida recibe una entrada de los nodos de las capas previas y calculan un valor de salida para la siguiente capa. La operación de un MLP con una capa oculta y una capa de salida lineal se puede expresar matemáticamente de la siguiente forma:

$$y_k = \sum_{j=1}^{N_H} w_{kj}\ \sigma\left(\sum_{i=1}^{N_I} w_{ji}x_i \pm \theta_j\right) \pm \theta_k \tag{2.6}$$

donde x_i representa a cada una de a las N_I entradas de la red, N_H es el número de neuronas en la capa oculta, y_k a la $k-$ésima salida de la capa final, w_{ji} son los pesos de la capa oculta y θ_j sus umbrales, w_{kj} los pesos de la capa de salida y θ_k sus umbrales. La estructura de un MLP con $k = 1$ se presenta gráficamente en la Figura 2.5. Esta es la arquitectura más común de MLP (Haykin, 2007).

Un resultado importante con respecto a los modelos MLP es el denominado *teorema de aproximación universal* (Cybenko, 1989) que establece que dado una suficiente cantidad de neuronas en la capa oculta, una red neuronal MLP puede aproximar cualquier función continua limitada a cualquier precisión especificada;

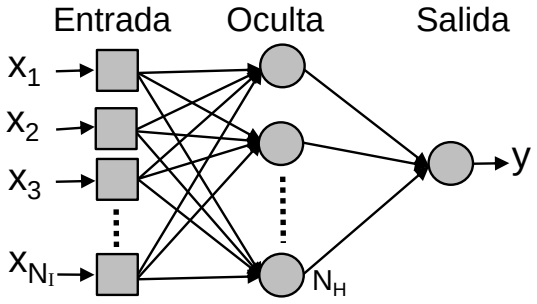

Figura 2.5: Modelo neuronal Perceptron Multi-Capa (MLP).

en otras palabras, siempre existe una red neuronal MLP de tres capas que puede aproximar cualquier función no lineal arbitraria, continua y multidimensional, para cualquier precisión deseada, siempre que el modelo tenga suficiente unidades neuronales (Hornik y otros, 1989). Sin embargo, cabe mencionar que el teorema no dice que una red de una sola capa es óptima en el sentido del tiempo de aprendizaje o la facilidad de aplicación; es más, el teorema no da indicaciones sobre el número requerido de unidades ocultas necesarias para conseguir el grado deseado de precisión (Burrascano y otros, 1999). La determinación de la cantidad suficiente de neuronas ocultas se logra, en la mayoría de los casos, mediante un procedimiento de prueba y error.

En un principio, las RNAs fueron utilizadas casi exclusivamente para modelar funciones estáticas de las variables de entrada solamente. Sin embargo, han sido utilizadas también para modelar efectivamente sistemas dinámicos (Stegmayer y Chiotti, 2009). Es decir, una red MLP puede ser utilizada también para modelar el comportamiento de la relación entre dos señales que varían en el tiempo. De acuerdo a la habilidad para modelar esta relación dentro de una cantidad limitada de iteraciones o épocas de entrenamiento con un bajo error, puede ser posible detectar una relación entre variables representadas por dos series temporales. La Figura 2.6 muestra un modelo MLP que tiene en cuenta explícitamente retardos de tiempo en la señal de entrada, es decir, no se incluye solamente el valor de la señal de entrada en el instante t, sino que también se consideran los valores en los instantes previos,

$$y_k(t) = \sum_{j=1}^{N_H} w_{kj}\ \sigma\left(\sum_{i=1}^{N_I} w_{ji} x_i(t-\tau) \pm \theta_j\right) \pm \theta_k \tag{2.7}$$

para τ entre 0 y T.

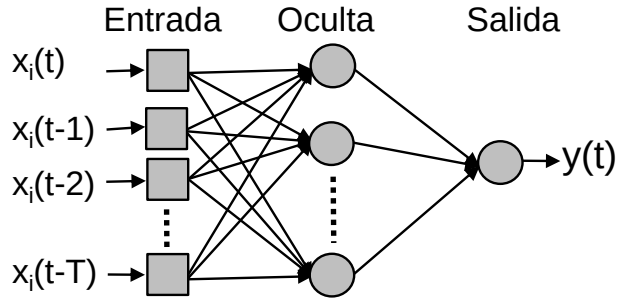

Figura 2.6: Modelo MLP con retardos temporales en la entrada.

Las redes de tipo MLP han sido aplicadas con éxito para la resolución de problemas variados y complejos, entrenándolas de modo supervisado con el algoritmo conocido como de retroprogación de los errores (del inglés original backpropagation), el cual se basa en una regla de corrección de errores. Este algoritmo se explica a continuación.

2.2.4. Aprendizaje por retropropagación de errores

El método de aprendizaje básico para las RNAs del tipo MLP se denomina de retropropagación de errores (backpropagation) (Rumelhart y otros, 1986). Se trata de un método de optimización por el descenso del gradiente de una función de error, con el cual se busca minimizar el error cuadrático medio (MSE, sigla en inglés de Mean Square Error) de la salida calculada por la red neuronal. El proceso de entrenamiento de una red con este algoritmo involucra tres fases: la fase del cómputo hacia adelante con los patrones de entrenamiento de entrada, el cálculo del error cometido y la propagación hacia atrás de ese error con el ajuste correspondiente de los pesos (y umbrales) del modelo.

Una vez que se ha aplicado un patrón a la entrada de la red como estímulo, éste se propaga desde la primera capa a las siguientes, hasta generar una salida. La señal de salida se compara con la salida deseada y se calcula una señal de error para cada una de las variables de salida. Las señales de error se propagan hacia atrás, partiendo de la capa de salida, hacia todas las neuronas de la capa oculta que contribuyen directamente a la salida. Sin embargo las neuronas de la capa oculta sólo reciben una fracción de la señal total del error, basándose aproximadamente en la contribución relativa que haya aportado cada neurona a la salida original. Este proceso se repite, capa por capa, hasta que todas las

neuronas de la red hayan recibido una señal de error que describa su contribución relativa al error total. Basándose en la señal de error percibida, se actualizan los pesos de conexión de cada neurona, para hacer que la red converja hacia un estado que permita aprender correctamente todos los patrones de entrenamiento.

La importancia de este proceso consiste en que, a medida que se entrena la red, las neuronas aprenden a reconocer distintas características del espacio total de entrada. Después del entrenamiento, cuando se le presente un patrón arbitrario de entrada que contenga ruido o que esté incompleto, las neuronas responderán con una salida activa si la nueva entrada contiene un patrón que se asemeje a aquella característica que las neuronas individuales hayan aprendido a reconocer durante su entrenamiento.

El algoritmo de retropropagación de errores para redes multicapa realiza su labor de actualización de pesos y umbrales en base al MSE. La red, al trabajar con aprendizaje supervisado, necesita un conjunto de datos de entrenamiento que le describa cada entrada y su valor de salida esperado o target t_k. El algoritmo debe ajustar los parámetros de la red para minimizar el error cuadrático medio de todos los patrones de entrenamiento, y funciona del siguiente modo.

El error para la neurona de salida j en la iteración o época de entrenamiento n se define como

$$e_j(n) = t_j(n) - y_j(n). \tag{2.8}$$

Se define el valor instantáneo del error cuadrático para la neurona j como $\frac{1}{2}e_j^2(n)$. El valor instantáneo $E(n)$ de la suma de los errores cuadráticos se obtiene sumando $\frac{1}{2}e_k^2(n)$ sobre todas las neuronas de la capa de salida N_O

$$E(n) = \frac{1}{2}\sum_{j\in N_O} e_j^2(n). \tag{2.9}$$

Por lo tanto, el error cuadrático medio se puede calcular como la suma anterior normalizada con respecto al tamaño N del conjunto de entrenamiento

$$MSE = \frac{1}{N}\sum_{n=1}^{N}\frac{1}{2}\sum_{j\in N_O} e_j^2(n). \tag{2.10}$$

La suma de errores cuadráticos y el MSE son función de los parámetros libres (pesos y umbrales) de la red. Para un conjunto de entrenamiento dado, el

MSE representa una medida del desempeño de la red en el aprendizaje de esos datos. El objetivo del aprendizaje es ajustar los parámetros libres de la red, de modo tal de minimizar el MSE. Para realizar esta minimización, se considera un método simple de entrenamiento en el que los pesos son actualizados patrón a patrón. Los ajustes a los pesos se realizan en concordancia con el respectivo error computado para cada patrón pesentado en la red. El promedio aritmético de estos cambios de los pesos individuales sobre el conjunto de entrenamiento es, por lo tanto, una estimación del verdadero cambio que resultaría de modificar los pesos basándose en la minimización de la función de costo MSE sobre el conjunto de entrenamiento completo (Haykin, 2007).

El algoritmo de retropropagación de errores aplica una corrección $\Delta w_{ji}(n)$ al peso sináptico $w_{ji}(n)$, el que es proporcional al gradiente instantáneo $\frac{\partial E(n)}{\partial w_{ji}(n)}$. De acuerdo a la regla de la cadena, es posible expresar este gradiente como

$$\frac{\partial E(n)}{\partial w_{ji}(n)} = \frac{\partial E(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial u_j(n)} \frac{\partial u_j(n)}{\partial w_{ji}(n)} \tag{2.11}$$

donde $y_j(n) = \sigma(u_j(n))$ y $u_j(n) = \sum_{i=1}^{N_I} w_{ji}(n) x_i(n)$. Esto resulta en

$$\frac{\partial E(n)}{\partial w_{ji}(n)} = -e_j(n)\sigma'(u_j(n))x_i(n). \tag{2.12}$$

Entonces, la corrección $\Delta w_{ji}(n)$ aplicado a $w_{ji}(n)$ es definido por la regla delta

$$\Delta w_{ji}(n) = -\eta \frac{\partial E(n)}{\partial w_{ji}(n)}, \tag{2.13}$$

donde η es una constante que determina la tasa o velocidad de aprendizaje. El uso del signo menos en (2.13) denota el descenso por el gradiente del error en el espacio de los pesos. Uniendo (2.12) y (2.13) se tiene, entonces

$$\Delta w_{ji}(n) = \eta e_j(n)\sigma'(u_j(n))x_i(n). \tag{2.14}$$

Por lo tanto, y en resumen, el gradiente local para una neurona j es igual al producto de la correspondiente señal de error $e_j(n)$ y la derivada de la función de activación de las neuronas asociadas a la capa a la cual pertenece la neurona. En estas ecuaciones puede notarse que un factor clave involucrado en el cálculo del ajuste del peso $\Delta w_{ji}(n)$ es la señal de error a la salida de la neurona. En este contexto, es posible identificar dos casos distintos, dependiendo de dónde

está localizada la neurona en la red. En el primer caso, la neurona j es un nodo de salida. Esto es fácil de manejar porque cada nodo de salida de la red es abastecido con una respuesta deseada de sí mismo, lo que hace sencillo de calcular el error asociado. Por lo tanto, es posible usar (2.8) para calcular la señal de error $e_j(n)$ asociada con esta neurona; y con ésta, se calcula el gradiente local (2.14). En el segundo caso, la neurona j es un nodo oculto, por lo que no hay una respuesta deseada específica para esa neurona. En consecuencia, la señal de error para esta neurona oculta debería determinarse recursivamente en términos de las señales de error para todas las neuronas con las que está directamente conectada.

Considerando (2.14), es posible ahora definir el gradiente local para una neurona oculta j como

$$\Delta w_{ji}(n) = -\eta \frac{\partial E(n)}{\partial y_j(n)} \sigma'(u_j(n)) x_i(n), \tag{2.15}$$

donde ahora la neurona j es una neurona oculta, y

$$\frac{\partial E(n)}{\partial y_j(n)} = \sum_k e_k(n) \frac{\partial e_k(n)}{\partial y_j(n)}. \tag{2.16}$$

Ahora hay que notar que $e_k(n) = t_k(n) - y_k(n)$ es el error de una neurona de salida k, y por lo tanto $y_k(n) = \sigma(u_k(n))$ y $u_k(n) = \sum_{j=1} w_{kj}(n) y_j(n)$.

Usando la regla de la cadena para la derivada parcial $\frac{\partial e_k(n)}{\partial y_j(n)}$ se reescribe la ecuación anterior y se obtiene

$$\frac{\partial E(n)}{\partial y_j(n)} = -\sum_k e_k(n) \sigma'(u_k(n)) w_{kj}(n). \tag{2.17}$$

Finalmente, se obtiene la regla de actualización de pesos en la capa oculta como

$$\Delta w_{ji}(n) = \eta \sum_k e_k(n) \sigma'(u_k(n)) w_{kj}(n) \sigma'(u_j(n)) x_i(n). \tag{2.18}$$

2.3. Modelo de serie de Volterra

En esta sección se presentan un conjunto de fórmulas para la extracción de los kernels de Volterra a partir de una red neuronal tipo MLP entrenada, independientemente de la topología del modelo neuronal, el número de variables involucradas en el problema y la nolinealidad que presenta el problema. En primer lugar se explica la serie de Volterra y sus características, para luego poder

demostrar la equivalencia que existe entre ésta y una red neuronal entrenada, a través de la extracción de los kernels.

2.3.1. Serie de Volterra

La serie de Volterra y el teorema de Volterra fueron desarrollados en 1887 por Vito Volterra. Es un modelo que permite representar comportamiento dinámico y no lineal, utilizado con frecuencia en la identificación de sistemas (Volterra, 1959). Por ejemplo, si se considera a una aproximación de tercer orden como suficiente para describir el sistema bajo estudio, un modelo de Volterra de tercer orden y de tiempo discreto relacionaría las entradas $x(t)$ y la salida $y(t)$ del siguiente modo

$$\begin{aligned} y(t) &= h_0 + \\ &\sum_{k=0}^{\infty} h_1(k)x(t-k) + \\ &\sum_{k_1=0}^{\infty}\sum_{k_2=0}^{\infty} h_2(k_1,k_2)x(t-k_1)x(t-k_2) + \\ &\sum_{k_1=0}^{\infty}\sum_{k_2=0}^{\infty}\sum_{k_3=0}^{\infty} h_3(k_1,k_2,k_3)x(t-k_1)x(t-k_2)x(t-k_3). \end{aligned} \tag{2.19}$$

donde h_0, $h_1(.)$, $h_2(.)$ y $h_3(.)$ son denominados *kernels de Volterra*. Como puede observarse, éstos se combinan con las entradas para obtener la salida del sistema. Se impone generalmente la condición de unicidad de los kernels asumiendo que son simétricos, es decir, para cualquier orden $p \geq 2$, $h_p(k_i, \cdots, k_p) = h_p(\pi(k_i, \cdots, k_p))$, donde $\pi(.)$ es cualquier permutación de $(k_1, \cdots, k_p)$. Puede demostrarse que los kernels de un sistema pueden asumirse simétricos sin pérdida de generalidad (Kibangou y Favier, 2009).

2.3.2. Cálculo de los kernels de Volterra

En (Stegmayer y Chiotti, 2009) se presentaron un conjunto de fórmulas para la extracción de los kernels de Volterra de una red neuronal MLP entrenada con un algoritmo clásico de retropropagación del error (Marquardt, 1963). Las ecuaciones permiten el cálculo a partir de una MLP de tres capas con neuronas

ocultas con función de activación tangente hiperbólica, independientemente del número de variables y la nolinealidad del problema. El modelo neuronal debe entrenarse utilizando los datos de entrenamiento disponibles y, luego de esto, se obtienen los kernels de Volterra a partir de los parámetros de la red entrenada, como se mostrará a continuación.

El procedimiento comienza expandiendo la siguiente salida de un modelo MLP de dos variables de entrada (N_I=2) con entradas retardadas en el tiempo $t-k$ y una sola variable de salida

$$y(t) = \theta + \sum_{j=1}^{N_H} w_j \tanh\left(\theta_j + \sum_{i=1}^{N_I} w_{ji} x_i(t-k)\right), \tag{2.20}$$

para k de 0 a T, como una serie de Taylor alrededor de los valores de los umbrales de los nodos ocultos, para lo cual se calculan las derivadas de las funciones de activación de las neuronas ocultas con respecto a los umbrales según se propone en (Stegmayer y Chiotti, 2009). Desarrollando y agrupando por factor común se obtiene una ecuación que, por ejemplo, para la MLP anterior con dos variables de entrada x_1 y x_2, y considerando solamente hasta las derivadas de tercer orden ($d = 3$), tiene la forma

$$
\begin{aligned}
y(t) \;=\; & \theta + \sum_{j=1}^{N_H} w_j \tanh(\theta_j) + \\
& \left[\sum_{j=1}^{N_H} w_j w_{j1} \left(\frac{\partial \tanh}{\partial \theta_j}\right)\right] x_1(t-k) + \\
& \left[\sum_{j=1}^{N_H} w_j w_{j2} \left(\frac{\partial \tanh}{\partial \theta_j}\right)\right] x_2(t-k) + \\
& \left[\sum_{j_1=1}^{N_H}\sum_{j_2=1}^{N_H} w_{j_1} w_{j_2 1} w_{j_2 1} \left(\frac{\frac{\partial^2 \tanh}{\partial \theta_j^2}}{2!}\right)\right] x_1^2(t-k) + \\
& \left[\sum_{j_1=1}^{N_H}\sum_{j_2=1}^{N_H} w_{j_1} w_{j_2 2} w_{j_2 2} \left(\frac{\frac{\partial^2 \tanh}{\partial \theta_j^2}}{2!}\right)\right] x_2^2(t-k) + \\
& \left[\sum_{j=1}^{N_H} w_j w_{j1} w_{j2} \left(\frac{\frac{\partial^2 \tanh}{\partial \theta_j^2}}{2!}\right)\right] x_1(t-k)x_2(t-k) + \cdots + \\
& \left[\sum_{j_1=1}^{N_H}\sum_{j_2=1}^{N_H} w_{j_1} w_{j_2 1} w_{j_2 1} w_{j_2 1} \left(\frac{\frac{\partial^3 \tanh}{\partial \theta_j^3}}{3!}\right)\right] x_1^3(t-k) + \\
& \left[\sum_{j_1=1}^{N_H}\sum_{j_2=1}^{N_H} w_{j_1} w_{j_2 2} w_{j_2 2} w_{j_2 2} \left(\frac{\frac{\partial^3 \tanh}{\partial \theta_j^3}}{3!}\right)\right] x_2^3(t-k) + \\
& \left[\sum_{j_1=1}^{N_H}\sum_{j_2=1}^{N_H}\sum_{j_3=1}^{N_H} w_{j_1} w_{j_2 1} w_{j_2 1} w_{j_3 2} \left(\frac{\frac{\partial^3 \tanh}{\partial \theta_j^3}}{3!}\right)\right] x_1^2(t-k)x_2(t-k) + \\
& \left[\sum_{j_1=1}^{N_H}\sum_{j_2=1}^{N_H}\sum_{j_3=1}^{N_H} w_{j_1} w_{j_2 1} w_{j_3 2} w_{j_3 2} \left(\frac{\frac{\partial^3 \tanh}{\partial \theta_j^3}}{3!}\right)\right] x_1(t-k)x_2^2(t-k). \quad (2.21)
\end{aligned}
$$

Si se compara (2.21) con el modelo doble de series de Volterra (representación para dos variables de entrada) (Nam y Powers, 2003; Pedro y otros, 2003), parcialmente reproducido a continuación,

$$
\begin{aligned}
y(t) \;=\; & h_0 + \sum_{k=0}^{+\infty} h_1(k)x_1(t-k) + \\
& \sum_{k=0}^{+\infty} h_1(k)x_2(t-k) + \\
& \sum_{k=0}^{+\infty} h_2(k,k)x_1^2(t-k) + \\
& \sum_{k=0}^{+\infty} h_2(k,k)x_2^2(t-k) + \\
& \sum_{k=0}^{+\infty} h_2(k,k)x_1(t-k)x_2(t-k) + \\
& \sum_{k=0}^{+\infty} h_3(k,k,k)x_1^3(t-k) + \\
& \sum_{k=0}^{+\infty} h_3(k,k,k)x_2^3(t-k) + \\
& \sum_{k=0}^{+\infty} h_3(k,k,k)x_1^2(t-k)x_2(t-k) + \\
& \sum_{k=0}^{+\infty} h_3(k,k,k)x_1(t-k)x_2^2(t-k), \qquad (2.22)
\end{aligned}
$$

los kernels de Volterra h_n pueden fácilmente ser reconocidos como los términos entre corchetes en la ecuación obtenida a partir del desarrollo de la salida de la MLP (2.21). Es decir, los kernels son calculables a partir de los parámetros (pesos y umbrales) de la MLP entrenada con los datos del sistema bajo estudio. De esta comparación se pueden obtener fórmulas generales que permiten el cálculo de un kernel de Volterra de cualquier orden utilizando los pesos y los valores umbrales de las neuronas ocultas de una MLP con cualquier topología (cualquier número de neuronas de entrada, cualquier número de neuronas ocultas y cualquier función de activación $\sigma(x)$). Se muestran a continuación las correspondientes hasta el tercer orden.

$$h_0 = \theta_0 + \sum_{j=1}^{N_H} w_j \sigma(\theta_j) \tag{2.23}$$

$$h_1(k) = \sum_{j=1}^{N_H} w_j w_{j1} \left(\frac{\partial \sigma}{\partial \theta_j} \right) \tag{2.24}$$

$$h_2(k,k) = \sum_{j=1}^{N_H} w_j w_{j1} w_{j2} \frac{\left(\frac{\partial^2 \sigma}{\partial \theta_j^2} \right)}{2!} \tag{2.25}$$

$$h_3(k,k,k) = \sum_{j=1}^{N_H} w_j w_{j1} w_{j2} w_{j2} \frac{\left(\frac{\partial^3 \sigma}{\partial \theta_j^3} \right)}{3!} \tag{2.26}$$

El siguiente grupo de ecuaciones puede utilizarse cuando las neuronas en la capa oculta poseen en particular la función de activación tangente hiperbólica. En las fórmulas, se ha desarrollado la derivada de la tangente hiperbólica, donde $\tanh = \tanh(\theta_j)$.

$$h_0 = b_0 + \sum_{j=1}^{N_H} w_j \tanh \tag{2.27}$$

$$h_1(k) = \sum_{j=1}^{N_H} w_j w_{j1} \left(1 - \tanh^2\right) \tag{2.28}$$

$$h_2(k,k) = \sum_{j=1}^{N_H} w_j w_{j1} w_{j2} \frac{\left(-2\tanh + 2\tanh^3\right)}{2!} \tag{2.29}$$

$$h_3(k,k,k) = \sum_{j=1}^{N_H} w_j w_{j1} w_{j2} w_{j2} \frac{\left(-2 + 8\tanh^2 - 6\tanh^4\right)}{3!} \tag{2.30}$$

2.4. Medidas de desempeño utilizadas.

En esta sección se explican las medida de desempeño que serán utilizadas para evaluar el desempeño de los modelos propuestos en esta tesis. En primer lugar se presentan las tasas de reconocimiento y de ahorro de espacio. Luego, se introducen los conceptos de Precisión y Sensitividad.

2.4.1. Tasas de reconocimiento y ahorro de espacio.

En problemas de clasificación, la fuente primaria de las mediciones de rendimiento es la precisión global de un clasificador estimada a través de la la tasa de clasificación o reconocimiento RR (Duda y Hart, 2003). Ésta se puede calcular como

$$RR = \frac{P_{CC}}{P_{AC}} \tag{2.31}$$

donde P_{CC} es el número de patrones de datos correctamente clasificados y P_{AC} es el número total de los patrones a clasificar.

Para la medición de la compresión, la relación de compresión de datos se puede utilizar para cuantificar la reducción en el tamaño de representación de datos producido por un algoritmo de compresión. Sin embargo, en su lugar, se presenta aquí la medida de ahorro de espacio en cuanto a la cantidad de parámetros, definido como la reducción de tamaño en relación con el espacio no comprimido (Salomon, 2007), a menudo descrito como un porcentaje, que da una mejor idea de la potencia de compresión. Para estimar cuál es el nivel de compresión obtenido, se calcula la tasa de ahorro de espacio SS como

$$SS = 1 - \frac{P_{nuevo}}{P_{original}} \tag{2.32}$$

siendo ésta una relación entre P_{nuevo} que es la cantidad de parámetros que se necesita para obtener una salida del nuevo modelo que se propone, y $P_{original}$ que es la cantidad de parámetros de cada uno de los modelos MLP correspondientes (los pesos y los umbrales). Para ambos casos, RR y SS, cuanto mayor sea la tasa, mejor será el resultado.

En particular, para un modelo MLP, la cantidad de parámetros se calcula como sigue:

$$P_{MLP} = N_I \times N_H + N_{BH} + N_H \times N_O + N_{BO}, \tag{2.33}$$

siendo N_I, N_H y N_O el número de neuronas en las capas de entrada, oculta y salida, respectivamente, y N_{BH} and N_{BO} son el número de umbrales para cada una de las neuronas pertenecientes a las capas oculta y de salida, respectivamente. En un arreglo de modelos MLP, estas medidas están afectadas por la longitud a

del arreglo. Así, la cantidad de parámetros necesarios para representar un modelo aMLP se calcula de esta manera:

$$P_{aMLP} = a \times P_{MLP}. \tag{2.34}$$

2.4.2. Precisión y Sensitividad

En problemas de clasificación binaria donde la variable de salida y puede tomar sólo dos valores, por ejemplo 0 o 1 (negativo o positivo), la comparación entre el valor predicho por una función determinada y la verdadera salida puede conducir a alguna de las siguientes cuatro situaciones:

1. Verdadero Positivo (VP): predicción correcta, el valor verdadero es 1, y la predicción también devuelve 1.

2. Verdadero Negativo (VN): predicción correcta, el valor verdadero es 0, y la predicción también devuelve 0.

3. Falso Positivo (FP): predicción incorrecta, la predicción devuelve 1, y en realidad el valor verdadero es 0.

4. Falso Negativo (FN): predicción incorrecta, la predicción devuelve 0, y en realidad el valor verdadero es 1.

Si se sumariza la cantidad de veces que ocurre cada una de las cuatro alternativas es usual con esos valores construir una matriz de confusión. A partir de los valores consignados en la matriz de confusión es posible definir distintas medidas para evaluar un clasificador, entre las cuales se encuentran la *precisión* y la *sensitividad*.

La *Precisión* es la proporción de resultados positivos (tanto los verdaderos positivos como los negativos) en la población de resultados, es decir, mide la proporción de ejemplos correctamente clasificados. Si la cantidad de ejemplos es lo suficientemente grande, puede considerarse la precisión como un estimador de la probabilidad de que un ejemplo sea correctamente clasificado a futuro. La expresión para el cálculo de la precisión es:

$$Precisión = \frac{VP + VN}{VP + VN + FP + FN}. \tag{2.35}$$

Por su parte, la *Sensitividad* refiere a la habilidad de identificar resultados positivos, es decir, mide la proporción de ejemplos con etiqueta positiva que han

sido correctamente clasificados, lo que es equivalente a medir la exactitud del clasificador para los ejemplos positivos. La expresión de cálculo es:

$$Sensitividad = \frac{VP}{VP + FN}. \tag{2.36}$$

2.5. Resumen

En este capítulo se introdujeron las Redes Neuronales Artificiales que pueden utilizarse para modelar problemas de clasificación. Luego, se presentó el modelo de series de Volterra y el proceso mediante el cual se calculan los kernels de la serie a partir de una red neuronal entrenada. La última parte del capítulo mostró las medidas de performances que se utilizan para medir los resultados.

CAPÍTULO 3

Compresión de clasificadores neuronales con el modelo V-NN

3.1. Introducción

En este capítulo, se explica en detalle el modelo red neuronal de Volterra (V-NN, de su sigla en inglés Volterra-Neural Network) propuesto, y se muestran los algoritmos para su cálculo. También se presenta cómo este modelo puede considerarse como una versión comprimida de un clasificador basado en una red neuronal tradicional, y de qué manera se utiliza para tareas de clasificación (Sección 3.3). Se detallan, además, las medidas utilizadas para medir el desempeño del modelo propuesto, como así también una nueva medida para seleccionar el mejor modelo de compromiso posible, mediante el análisis de las posibles soluciones obtenidas con la aplicación de V-NN (Sección 3.4).

El modelo y los algoritmos propuestos en este capítulo serán probados y validados en base a conjuntos de datos reconocidos en el área. La capacidad de compresión del modelo V-NN será probada inicialmente con el problema de la base de datos Iris (Sección 3.5). Luego, la compresión de un arreglo de clasificadores es probada sobre dos bases de datos bien conocidas en la literatura: reconocimiento de letras y de dígitos manuales escritos con lapicera (Sección 3.6). Por último, se extiende la evaluación del modelo aplicándolo sobre un problema real de reconocimiento de rostros, utilizando dos bases de datos independientes (Sección 3.7).

Del estudio realizado en este capítulo han surgido publicaciones en un congreso internacional con referato (Rubiolo y otros, 2010), en un congreso nacional con referato (Rubiolo, 2012), y en una revista indexada con Factor de Impacto

2012 de 1.168 (Rubiolo y otros, 2013b).

3.2. Compresión de modelos neuronales

El propósito de la compresión de modelos es encontrar un modelo compacto que aproxime una función aprendida, por ejemplo, por un clasificador. Más aún, es deseable alcanzar esto sin una pérdida significativa de desempeño (Buciluǎ y otros, 2006b). A menudo los modelos que mejor desempeño poseen, y que utilizan aprendizaje supervisado, son combinaciones de clasificadores grandes y complejos. Sin embargo, en algunas situaciones, para un clasificador no es suficiente con ser altamente preciso, sino que también debe satisfacer requerimientos relacionados al tiempo de ejecución, el espacio de almacenamiento y un limitado poder computacional (Zhang y Wangmeng, 2007), como por ejemplo el de los teléfonos celulares actuales y tablets.

Como ya se ha mencionado en la Introducción (Capítulo 1), los modelos neuronales del tipo Perceptrón Multicapa son generalmente considerados como modelos de clasificación muy potentes. Sin embargo, con el objetivo de aprender mejor los datos de entrenamiento y mejorar su capacidad de clasificación, fácilmente se convierten en grandes modelos, ya sea con sólo agregar más neuronas en la capa oculta o más variables de entrada. Esto implica directamente incrementar la cantidad de parámetros (pesos y umbrales) en el modelo.

En la literatura se encuentran diversas propuestas para reducir el tamaño de una red neuronal entrenada. En la mayoría de ellas se requiere la elección inicial de una estructura neuronal y luego se eliminan conexiones entre neuronas. Mediante la eliminación de pesos que no son representativos o importantes para una red, se espera mejorar su desempeño en cuanto a la capacidad de generalización, utilizar menos cantidad de parámetros para el entrenamiento, y aumentar la velocidad de aprendizaje.

Un método muy popular es el de "poda" ("pruning", en inglés) de una red, en el que se obtiene una red neuronal más pequeña mediante la reducción de la cantidad de nodos o conexiones que ésta tenga (Kavzoglu y Mather, 1998) (Siebel y otros, 2009). Existen dos algoritmos clásicos de poda de pesos de una MLP entrenada, considerando a ésto como una forma de compresión.

El primer algoritmo, OBD (Cun y otros, 1990) (sigla de su nombre en inglés

Optimal Brain Damage), es una técnica para la reducción del tamaño de una red neuronal entrenada, mediante la selección de los pesos a eliminar. Para esto, encuentra los pesos de menor impacto, es decir, aquellos pesos cuyo valor, si es llevado a cero, tienen menos influencia en el error de entrenamiento. Para poder calcularlo, se utiliza la diagonal de la matriz Hessiana (matriz de las segundas derivadas parciales contínuas de una función f en un punto x).

El segundo algoritmo es el OBS (Hassibi y otros, 1993) (sigla de su nombre en inglés Optimal Brain Surgeon). La idea básica de este procedimiento es también estimar el aumento o disminución del error de entrenamiento cuando se eliminan pesos utilizando las derivadas de segundo orden del error. De esta manera, se construye un modelo local de la función error para evaluar el efecto de la eliminación de los pesos que no son demasiado importantes en una red neuronal entrenada. Lo que lo diferencia del algoritmo OBD es que, en este caso, se utiliza también toda la información contenida fuera de la diagonal de la matriz Hessiana.

Por un lado, sin embargo, al momento no se conoce técnica que permita la compresión de una red neuronal, independientemente del hecho de que haya sido podada o no. Se entiende por poda el obtener un modelo neuronal con menor cantidad de parámetros, es decir, una representación más compacta del mismo, con igual capacidad de generalización. Por otro lado, tal como se ha mostrado en el marco teórico, Sección 2.3.2, es posible extraer un modelo de series de Volterra a partir de los parámetros de una red neuronal entrenada. El modelo MLP es entrenado utilizando los datos disponibles y, luego de esto, se obtienen los kernels de la serie. Para lograrlo, se utilizan un conjunto de fórmulas para la extracción de los kernels de Volterra, independientemente de la topología del modelo neuronal, de las funciones de activación, del número de variables involucradas en el problema y de la no linealidad del sistema. Este procedimiento ha sido aplicado para reproducir el comportamiento no lineal y dinámico de los dispositivos de comunicación inalámbrica, en particular amplificadores de potencia en teléfonos celulares de primera y segunda generación (Stegmayer y Chiotti, 2009).

3.3. El modelo neuronal de Volterra

En este capítulo de la tesis se propone un modelo para la compresión de modelos de clasificación basados en MLP llamado Red Neuronal de Volterra.

Es decir, se muestra cómo un modelo MLP que ha aprendido un problema de clasificación con cierta precisión, generalmente alta, puede comprimirse en un representación más compacta utilizando un modelo de Volterra y sus parámetros, llamados en este contexto pesos de Volterra. Esta nueva propuesta involucra una menor cantidad de parámetros, manteniendo, sin embargo, una alta precisión en la clasificación. Las próximas subsecciones presentan en detalle las partes necesarias para la construcción del modelo V-NN.

3.3.1. Obtención del modelo V-NN

En el marco teórico se ha explicado que existe una fuerte equivalencia entre una MLP y el modelo de series de Volterra. Así, recordemos a continuación la ecuación (2.20), correspondiente a la salida de una red MLP estándar con dos variables de entrada

$$y(t) = \theta + \sum_{j=1}^{N_H} w_j \tanh\left(\theta_j + \sum_{i=1}^{2} w_{ji} x_i(t)\right), \tag{3.1}$$

a partir de la cual, luego del desarrollo detallado en la Sección 2.3, se puede obtener la siguiente ecuación

$$
\begin{aligned}
y(t) \;=\; & \theta + \sum_{j=1}^{N_H} w_j \tanh(\theta_j) + \\
& \left[\sum_{j=1}^{N_H} w_j w_{j1} \left(\frac{\partial \tanh}{\partial \theta_j}\right)\right] x_1(t) + \\
& \left[\sum_{j=1}^{N_H} w_j w_{j2} \left(\frac{\partial \tanh}{\partial \theta_j}\right)\right] x_2(t) + \\
& \left[\sum_{j_1=1}^{N_H}\sum_{j_2=1}^{N_H} w_{j_1} w_{j_2 1} w_{j_2 1} \left(\frac{\frac{\partial^2 \tanh}{\partial \theta_j^2}}{2!}\right)\right] x_1^2(t) + \\
& \left[\sum_{j_1=1}^{N_H}\sum_{j_2=1}^{N_H} w_{j_1} w_{j_2 2} w_{j_2 2} \left(\frac{\frac{\partial^2 \tanh}{\partial \theta_j^2}}{2!}\right)\right] x_2^2(t) + \\
& \left[\sum_{j=1}^{N_H} w_j w_{j1} w_{j2} \left(\frac{\frac{\partial^2 \tanh}{\partial \theta_j^2}}{2!}\right)\right] x_1(t-k) x_2(t) + \cdots + \\
& \left[\sum_{j_1=1}^{N_H}\sum_{j_2=1}^{N_H} w_{j_1} w_{j_2 1} w_{j_2 1} w_{j_2 1} \left(\frac{\frac{\partial^3 \tanh}{\partial \theta_j^3}}{3!}\right)\right] x_1^3(t) + \\
& \left[\sum_{j_1=1}^{N_H}\sum_{j_2=1}^{N_H} w_{j_1} w_{j_2 2} w_{j_2 2} w_{j_2 2} \left(\frac{\frac{\partial^3 \tanh}{\partial \theta_j^3}}{3!}\right)\right] x_2^3(t) + \\
& \left[\sum_{j_1=1}^{N_H}\sum_{j_2=1}^{N_H}\sum_{j_3=1}^{N_H} w_{j_1} w_{j_2 1} w_{j_2 1} w_{j_3 2} \left(\frac{\frac{\partial^3 \tanh}{\partial \theta_j^3}}{3!}\right)\right] x_1^2(t) x_2(t) + \\
& \left[\sum_{j_1=1}^{N_H}\sum_{j_2=1}^{N_H}\sum_{j_3=1}^{N_H} w_{j_1} w_{j_2 1} w_{j_3 2} w_{j_3 2} \left(\frac{\frac{\partial^3 \tanh}{\partial \theta_j^3}}{3!}\right)\right] x_1(t) x_2^2(t). \qquad (3.2)
\end{aligned}
$$

Por otro lado, también se presentó la ecuación del modelo de serie de Volterra (2.22), para dos variables de entrada sin retardos temporales, tal como se muestra parcialmente reproducida en la siguiente ecuación:

$$
\begin{aligned}
y(t) \;=\; & h_0 + \sum h_1(k)x_1(t) + \\
& \sum h_1(k)x_2(t) + \\
& \sum h_2(k,k)x_1(t)^2 + \\
& \sum h_2(k,k)x_2(t)^2 + \\
& \sum h_2(k,k)x_1(t)x_2(t) + \\
& \sum h_3(k,k,k)x_1(t)^3 + \\
& \sum h_3(k,k,k)x_2(t)^3 + \cdots
\end{aligned} \tag{3.3}
$$

Comparando las dos ecuaciones mostradas anteriormente, es posible ver que los kernels de Volterra (h_i en (3.3)) son calculables a partir de los parámetros de MLP (entre corchetes en (3.2)).

Las ecuaciones (2.27) a (2.30) del marco teórico permiten el cálculo de un kernel de Volterra de hasta tercer orden, utilizando los pesos y los umbrales de las neuronas ocultas de una red neuronal que ha sido entrenada utilizando la función tangente hiperbólica. En cambio, para los modelos MLP utilizados en este capítulo, la función de activación utilizada en las neuronas de la capa oculta es la función sigmoidea. Por lo tanto, las siguientes ecuaciones permiten el cálculo de los kernels de Volterra de orden cero, uno, dos y tres, a partir de una MLP entrenada, considerando dos variables de entrada y una función de activación sigmoidea en las unidades ocultas:

$$h_0 = \theta + \sum_{j=1}^{N_H} w_j \frac{1}{(1+e^{-\theta_j})} \tag{3.4}$$

$$h_1(\cdot) = \sum_{j=1}^{N_H} w_j w_{j1} \frac{e^{-\theta_h}}{(1+e^{-\theta_h})^2} \tag{3.5}$$

$$h_2(\cdot) = \sum_{j=1}^{N_H} w_j w_{j1} w_{j2} \frac{\frac{e^{-\theta_h}(e^{-\theta_h}-1)}{(1+e^{-\theta_h})^3}}{2!} \tag{3.6}$$

$$h_3(\cdot) = \sum_{j=1}^{N_H} w_j w_{j1} w_{j2} w_{j2} \frac{\frac{-e^{-\theta_h}(-e^{-2\theta_h}+4e^{-1}-1)}{(1+e^{-\theta_h})^4}}{3!} \tag{3.7}$$

donde N_H es el número de neuronas ocultas, w_j y θ_j son el peso y el umbral asociados a una neurona sigmoidea en el capa oculta, respectivamente. Estas

fórmulas pueden extenderse fácilmente a kernels de cualquier orden n, tal como se muestra a continuación

$$h_n(\cdot) = \sum_{j=1}^{N_H} w_j w_{j1} \cdots w_{jn} \frac{\left(\frac{\partial^n \sigma}{\partial \theta_j}\right)}{n!}. \tag{3.8}$$

Para lograr una mejor comprensión del nuevo modelo propuesto, y lograr una correcta asociación de la notación de los parámetros, a lo que anteriormente se llamaba kernel de Volterra, se lo llamará *pesos de Volterra* de ahora en adelante. Por lo tanto, el kernel de orden cero que anteriormente se identificaba con h_0, de ahora en adelante se llamará peso de orden cero y se representará con $v^{(0)}$. De la misma manera, ocurrirá con los nuevos pesos de Volterra de orden uno, dos y tres: $v_i^{(1)} = h_1(\cdot)$, $v_{i,j}^{(2)} = h_2(\cdot)$ y $v_{i,j,k}^{(3)} = h_3(\cdot)$.

Teniendo en cuenta esto, y la forma que tiene el modelo de series de Volterra, se propone la topología del modelo Volterra-NN, la cual se muestra gráficamente en la Figura 3.1. En la misma puede observarse cómo las variables de entrada se combinan con los correspondientes pesos de Volterra para producir las salidas del modelo. Las cajas representan a las variables de entrada (en el ejemplo, x_1 y x_2) y las flechas que las unen simbolizan el producto con sus correspondientes pesos de Volterra. Los círculos actúan sumarizando todos los productos entre las variables de entrada y los pesos de Volterra, más la salida V-NN de orden n-1 ($s^{(n-1)}$). Como resultado, se obtiene una salida de Volterra de n^{mo} orden ($s^{(n)}$). Las diferentes combinaciones de productos cruzados entre las variables de entrada de ejemplo se muestran en la figura dentro de una caja rectangular. Este modelo V-NN podría aplicarse, de manera análoga, a cualquier número de entradas.

La salida del modelo V-NN de 1^{er} orden (s^1) se obtiene analíticamente agregando el peso de Volterra de orden 0 ($v^{(0)}$) al producto entre cada variable de entrada (x_1 y x_2) y su correspondiente peso de Volterra de 1^{er} orden:

$$s^{(1)} = v^{(0)} + v_1^{(1)}\ x_1 + v_2^{(1)}\ x_2. \tag{3.9}$$

La salida de segundo orden ($s^{(2)}$) del modelo V-NN se obtiene agregando $s^{(1)}$ junto al producto entre x_1^2, x_2^2, el producto cruzado $x_1 x_2$ y sus correspondientes pesos de Volterra de 2^{do} orden, resultando:

$$s^{(2)} = s^{(1)} + v_{1,1}^{(2)}\ x_1^2 + v_{2,2}^{(2)}\ x_2^2 + v_{1,2}^{(2)}\ x_1 x_2. \tag{3.10}$$

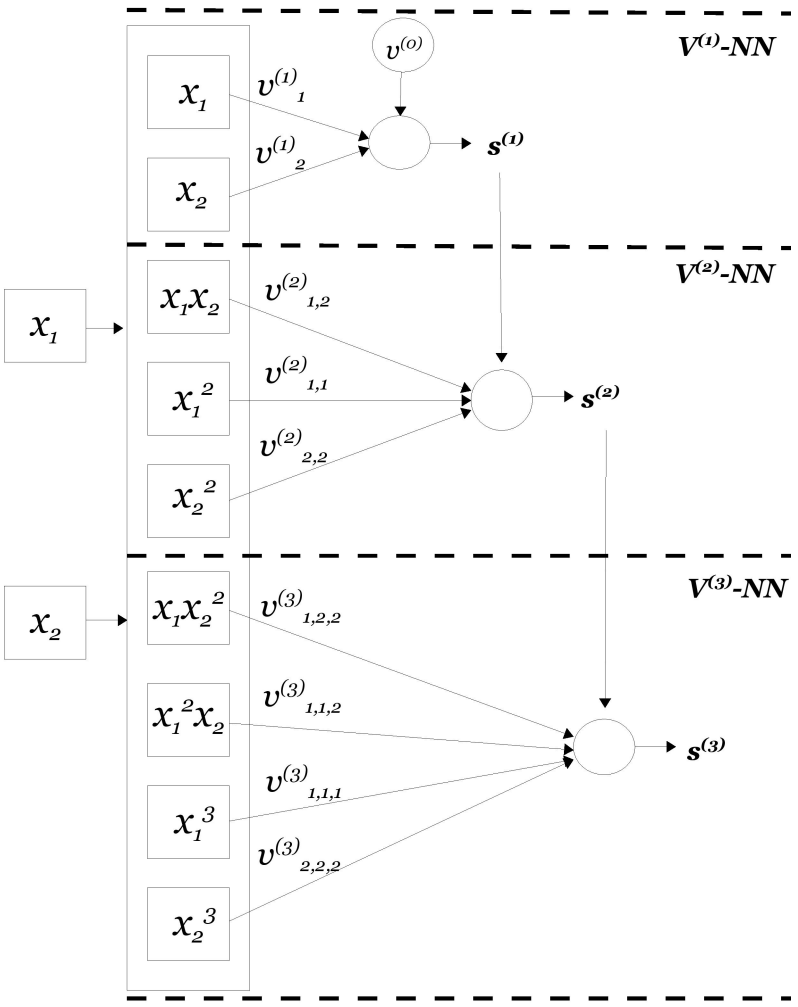

Figura 3.1: Ejemplo de la topología de un modelo Volterra-NN para dos variables de entrada y tres salidas V-NN.

La salida $s^{(3)}$ del modelo se obtiene de la siguiente manera:

$$s^{(3)} = s^{(2)} + v^{(3)}_{1,1,1}\ x_1^3 + v^{(3)}_{2,2,2}\ x_2^3 + v^{(3)}_{1,1,2}\ x_1^2 x_2 + v^{(3)}_{1,2,2}\ x_1 x_2^2. \tag{3.11}$$

Similarmente, se pueden calcular las salidas V-NN de orden más alto. Como se puede observar, cada modelo de Volterra de más alto orden incluye sus propios parámetros (pesos de Volterra) más los correspondientes a los órdenes más bajos.

El Algoritmo 1 muestra en detalle el procedimiento de extracción de los pesos de Volterra, en el cual es posible calcular los pesos de orden cero, uno, dos y tres. La entrada al algoritmo corresponde a los datos de entrenamiento y la salida da los pesos de Volterra. El primer paso consiste en entrenar el modelo MLP

Algoritmo 1: Extracción de los pesos de Volterra.

Data:
D: datos de entrenamiento
Results:
$v^{(0)}$: peso de Volterra de orden 0
$\mathbf{v}^{(1)}$: pesos de Volterra de 1^{er} orden
$\mathbf{v}^{(2)}$: pesos de Volterra de 2^{do} orden
$\mathbf{v}^{(3)}$: pesos de Volterra de 3^{er} orden

```
1  begin
2      M ← entrenar el modelo MLP con D
3      v^(0) ← calcular el peso de Volterra de orden cero a partir de M utilizando (3.4)
4      for 1 ≤ i ≤ N_I do
5          v_i^(1) ← calcular el peso de Volterra de primer orden a partir de M utilizando (3.5)
6      Asignar cada v_i^(1) extraído a v^(1)
7      for 1 ≤ i ≤ N_I do
8          for 1 ≤ j ≤ N_I do
9              v_{i,j}^(2) ← calcular el peso de Volterra de segundo orden a partir de M utilizando (3.6)
10     Asignar cada v_{i,j}^(2) extraído a v^(2)
11     for 1 ≤ i ≤ N_I do
12         for 1 ≤ j ≤ N_I do
13             for 1 ≤ k ≤ N_I do
14                 v_{i,j,k}^(3) ← calcular el peso de Volterra de tercer orden a partir de M utilizando (3.7)
15     Asignar cada v_{i,j,k}^(3) extraído a v^(3)
16 end
```

clasificador con el conjunto de datos de entrenamiento D, tal como se observa en la línea 2. A partir del modelo neuronal entrenado M, se calculan los pesos de Volterra. De acuerdo a (3.4), se obtiene el peso de Volterra orden cero (línea 3). De manera similar, aplicando (3.5), (3.6) and (3.7), es posible calcular los pesos de Volterra de orden uno, (línea 9), dos (línea 8) y tres (línea 7), respectivamente.

Sin embargo, los arreglos y ensambles de redes perceptrón multicapa han demostrado alcanzar significativamente mejores resultados en problemas de clasificación que los modelos simples (Aitkenhead y McDonald, 2003)(Dzeroski y Zenko, 2004)(Bianchini y otros, 2005)(Capello y otros, 2009)(Rahman y Verma, 2011). En un arreglo de MLPs (aMLP) hay un modelo MLP para cada clase c a identificarse, con c=1..C, siendo C el número total de clases. Por lo tanto, el modelo aMLP está formado por C redes tales como la que se muestra en la Figura 3.2.

Como ya se ha mostrado, el procedimiento de extracción de los pesos de Volterra permite construir una V-NN, que es una versión comprimida de un modelo MLP entrenado sobre una tarea de clasificación, manteniendo la misma tasa de reconocimiento que el modelo MLP original pero con menos parámetros. Dado que es posible obtener un modelo Volterra-NN de un único modelo MLP entrena-

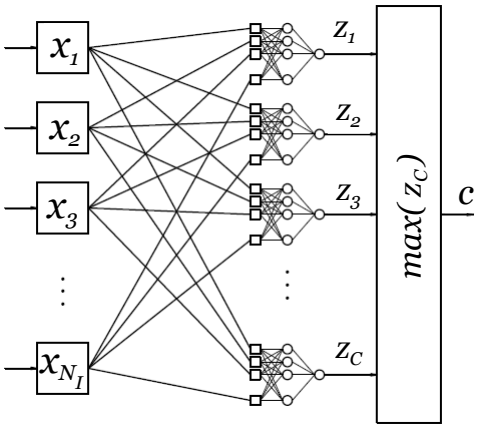

Figura 3.2: Modelo arreglo de MLPs (aMLP) para clasificación.

do, también es posible comprimir un arreglo de MLPs (aMLP) con V-NN. Esto se logra utilizando la misma metodología aplicada a construir un V-NN a partir de una sola MLP para obtener un arreglo de modelos V-NN (aV-NN) a partir de un aMLP, en el cual hay un V-NN por cada MLP del arreglo.

El Algoritmo 2 presenta en detalle la propuesta para el cálculo de las salidas de un aV-NN. Como se dijo anteriormente, este algoritmo está diseñado no sólo para obtener salidas de un modelo simple V-NN, sino que también este cálculo puede realizarse para un arreglo de modelos V-NN, considerando al modelo simple como un arreglo *1*V-NN. Es decir, el arreglo posee sólo un modelo. El algoritmo recibe un punto a clasificar desde dónde se determina el número de variables de entrada N_I, el número de elementos en el arreglo (por ejemplo, 1 para un modelo simple) y los pesos de Volterra para cada MLP miembro del arreglo, obtenidos utilizando el Algoritmo 1. La salida del Algoritmo 2 es un arreglo de salidas de Volterra de tercer orden ($V^{(3)} - NN$), pero es posible también obtener solamente la salida de primer orden ($V^{(1)} - NN$) (líneas 3 a 5) y la de segundo orden ($V^{(2)} - NN$) (líneas 7 a 10), proveyendo en consecuencia diferentes versiones comprimidas del clasificador original aMLP.

En el marco teórico, más precisamente en las ecuaciones (2.33) y (2.34), ha sido detallada la cantidad de parámetros que posee un modelo MLP y aMLP, respectivamente. A continuación se detallan las ecuaciones que deben utilizarse para obtener la cantidad de parámetros necesarios para poder representar una salida de V-NN de diferente orden, y cómo estas salidas efectivamente usan menos parámetros que una MLP.

Algoritmo 2: Cálculo de las salidas de Volterra-NN de 1^{ro}, 2^{do} y 3^{er} orden.

Data:
- $\mathbf{x}$: punto de entrada a clasificar
- K: número de elementos en el arreglo de MLPs
- $\mathbf{v}^{(0)}$: pesos de Volterra de orden 0 para el arreglo
- $\mathbf{v}^{(1)}$: arreglo de pesos de Volterra de 1^{er} orden
- $\mathbf{v}^{(2)}$: arreglo de pesos de Volterra de 2^{do} orden
- $\mathbf{v}^{(3)}$: arreglo de pesos de Volterra de 3^{er} orden

Results:
- $\mathbf{s}^{(1)}$: salidas $V^{(1)} - NN$ para el arreglo
- $\mathbf{s}^{(2)}$: salidas $V^{(2)} - NN$ para el arreglo
- $\mathbf{s}^{(3)}$: salidas $V^{(3)} - NN$ para el arreglo

```
1  begin
2      for cada elemento en el arreglo do
3          s^(1) ← v^(0)
4          for 1 ≤ i ≤ N_I do
5              s^(1) = s^(1) + v_i^(1) x_i
6          s^(2) ← s^(1)
7          for 1 ≤ i ≤ N_I do
8              for 1 ≤ j ≤ N_I do
9                  s^(2) = s^(2) + v_{i,j}^(2) x_i x_j
10         s^(3) ← s^(2)
11         for 1 ≤ i ≤ N_I do
12             for 1 ≤ j ≤ N_I do
13                 for 1 ≤ k ≤ N_I do
14                     s^(3) = s^(3) + v_{i,j,k}^(3) x_i x_j x_k
15         Asignar cada s^(1) calculado a s^(1)
16         Asignar cada s^(2) calculado a s^(2)
17         Asignar cada s^(3) calculado a s^(3)
18     Determinar los umbrales más bajo y más alto para cada elemento en el arreglo (clase)
19 end
```

De cada una de las MLP que forman parte de un arreglo de MLPs, se puede extraer una salida de Volterra-NN (de un orden particular): $s^{(1)}$ que incluye solamente los pesos de orden cero y de primer orden; $s^{(2)}$ incluye hasta el peso de Volterra de segundo orden; y $s^{(3)}$ corresponde a la salida de tercer orden. Las ecuaciones que se muestran a continuación detallan la cantidad de parámetros que se necesitan para obtener cada una de esas salidas.

Para la salida del modelo $V^{(1)} - NN$, tal como puede observarse en la ecuación 3.9, tenemos

$$P_{s^{(1)}} = N_I \tag{3.12}$$

donde $P_{s^{(1)}}$ es la cantidad de parámetros que se requieren para poder construir la salida del modelo V-NN de primer orden, y N_I es la cantidad de entradas al modelo. En este caso, el número de parámetros necesarios para construir $s^{(1)}$ es igual a la cantidad de entradas, dado que cada uno de los pesos de Volterra de primer orden multiplica a cada variable de entrada.

Con respecto a $s^{(2)}$, tal como puede observarse en la ecuación (3.10), el número de parámetros necesarios para construir $s^{(1)}$ se suman a la cantidad de pesos de Volterra de segundo orden. Cada uno de estos pesos se aplica al cuadrado de cada variable de entrada, más el producto cruzado entre ellas. Así, es posible pensar en una matriz de N_IxN_I en la que los elementos de la diagonal representan a cada variable al cuadrado, y los elementos por arriba o por abajo de la diagonal a los productos cruzados entre las mismas. De esta manera, es posible asociar a cada una de las celdas de esa matriz simétrica con un parámetro necesario para construir la salida, y la suma de los elementos de la triangular superior (o inferior) será la cantidad de parámetros de $s^{(2)}$. Para este cálculo se debe sumarizar la cantidad de elementos en la matriz (N_I^2), y restarle la cantidad de elementos que no se consideran (la triangular superior o inferior), según corresponda. A la cantidad de elementos totales de la matriz (N_I^2), hay que restarle los presentes en la diagonal (N_I^2-N_I) y luego dividirlo por 2. Por lo tanto, para obtener la salida del modelo de orden dos $V^{(2)}$-NN necesitamos la siguiente cantidad de parámetros $P_{s^{(2)}}$,

$$P_{s^{(2)}} = P_{s^{(1)}} + N_I^2 - \frac{N_I^2 - N_I}{2}. \tag{3.13}$$

Similarmente para $s^{(3)}$, según la ecuación (3.11), a la cantidad de pesos asociados con $s^{(2)}$ debe sumársele el producto de cada peso de tercer orden con la correspondiente variable de entrada, cuya cantidad es N_IxN_I. De esta manera, el número de parámetros necesarios para construir $V^{(3)}$-NN ($s^{(3)}$) es

$$P_{s^{(3)}} = P_{s^{(2)}} + N_I^2. \tag{3.14}$$

Como el modelo Volterra-NN puede aplicarse también a la compresión de un arreglo de MLP, en ese caso la salida será un arreglo de salidas V-NN de diferente orden. Esto significa que, por ejemplo, para cada MLP dentro del arreglo, se pueden obtener tres salidas V-NN de diferente orden. Para este caso, la cantidad de parámetros de los modelos V-NN deben redefinirse como

$$P_{as^{(i)}} = a \times P_{s^{(i)}}, \tag{3.15}$$

donde i = 1,2,3.

El cálculo de las cantidades de parámetros necesarios para la obtención de las salidas de diferentes órdenes de un modelo V-NN es necesario para poder computar la medida de ahorro de espacio SS introducida previamente en el marco

Tabla 3.1: Ejemplo de cálculo de parámetros para un clasificador MLP y sus correspondientes salidas de Volterra-NN de primer, segundo y tercer orden, considerando diferentes valores de parámetros. Se muestra además la tasa de ahorro de espacio SS para estos ejemplos.

N_I	2			3			4			5		
N_H	4			6			8			10		
N_O	1			1			1			1		
P_{MLP}	17			31			49			71		
	$s^{(1)}$	$s^{(2)}$	$s^{(3)}$	$s^{(1)}$	$s^{(2)}$	$s^{(3)}$	$s^{(1)}$	$s^{(2)}$	$s^{(3)}$	$s^{(1)}$	$s^{(2)}$	$s^{(3)}$
$P_{s^{(i)}}$	2	5	9	3	9	18	4	14	30	5	20	45
$SS_{s^{(i)}}$ [%]	88,24	70,59	47,06	90,32	70,97	41,94	91,84	71,43	38,78	92,96	71,83	36,62

teórico (sección 2.4.1). Considerando las ecuaciones (3.12) (3.13) (3.14) (3.15), el valor SS es posible calcularlo como

$$SS = 1 - \frac{P_{as^{(i)}}}{P_{aMLP}}. \tag{3.16}$$

En la Tabla 3.1 se muestra un ejemplo del cálculo de la cantidad de parámetros para diferentes modelos, y el correspondiente cálculo de la tasa de ahorro de espacio SS para esos valores. En la tres primeras filas se muestran la cantidad de neuronas en la capa de entrada N_I, oculta N_H y de salida N_O, para los modelos, considerando que en la capa oculta hay el doble de neuronas que en la capa de entrada, y una sola en la salida. En la cuarta fila se detalla la cantidad de parámetros necesarios para contruir un modelo MLP según la ecuación (2.33), para los valores de N_I, N_H y N_O. La sexta fila muestra la cantidad de parámetros necesarios para construir la salidas V-NN de orden 1, 2 y 3 según las ecuaciones (3.12), (3.13) y (3.14), respectivamente. Las última fila presenta el valor SS correspondiente a la comparación entre la cantidad de parámetros necesarios para obtener el modelo MLP (cuarta fila) y los necesarios para construir la salida de orden 1 ($s^{(3)}$), 2 ($s^{(3)}$) y 3 ($s^{(3)}$) (sexta fila) para cada modelo ejemplificado en cada columna.

3.3.2. El modelo Volterra-NN como clasificador

Esta subsección explica cómo el modelo Volterra-NN puede aplicarse a un problema de clasificación. El cálculo de la tasa de clasificación se realiza una vez que se establecen los límites de las clases sobre los datos de entrenamiento y son aplicados luego sobre los datos de testeo. Esto permite identificar las diferentes clases cuando se le presenta un dato desconocido al modelo a ser clasificado.

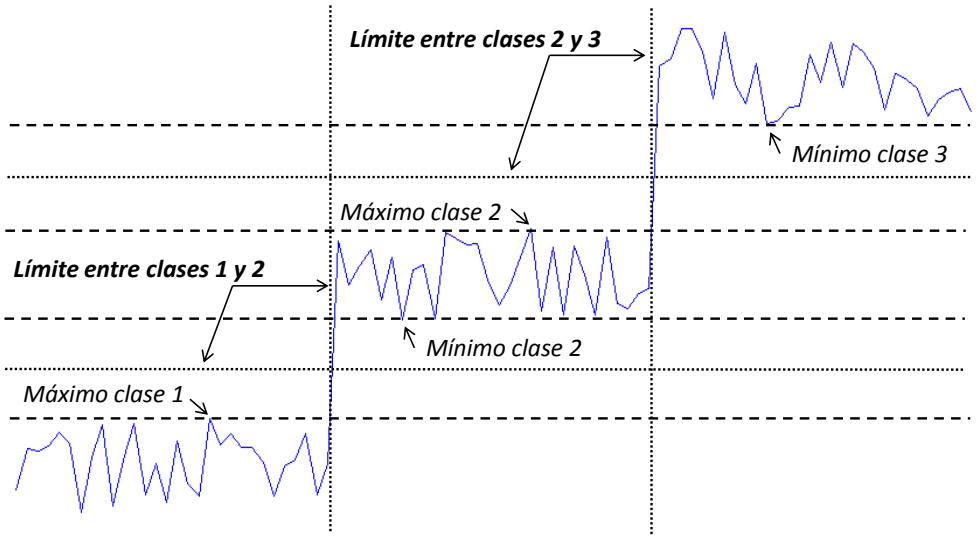

Figura 3.3: Cálculo de los límites entre tres clases usando los datos de entrenamiento.

Dos enfoques diferentes pueden considerarse para clasificar utilizando el modelo V-NN. En el primero, los datos de entrenamiento se muestran en forma ordenada, desde los ejemplos de la primer clase hasta la última. Es decir, si se utiliza un conjunto de datos con tres clases, en primer lugar se le muestran al modelo todos los datos correspondientes a la primer clase; luego los datos de la segunda clase y al final los de la tercera. De esta manera, la salida del modelo puede considerarse como una señal que posee tres niveles diferentes, con límites entre los niveles que se corresponden con los límites de las clases (Korenberg M. J. y Solomonc, 2001).

Para calcular estos límites, se miden los valores mínimos y máximos de cada clase para poder determinar el punto medio entre los cambios de nivel de la señal de salida. Estos cambios (umbrales) permiten identificar a qué clase pertenece un punto nuevo cuando se lo recibe para clasificar. Por ejemplo, un conjunto de datos puede tener tres niveles diferentes que representan las tres clases posibles a identificar. Esto se muestra en la Figura 3.3, donde las líneas punteadas verticales establecen el límite de cada clase. Para establecer los límites horizontales de cada clases, se detecta el valor mínimo y máximo de cada clase vecina. El límite se colocará en el nivel medio entre el valor máximo de la clase actual y mínimo de la clase siguiente, lo que puede observarse como líneas punteadas horizontales. Por

lo tanto, dependiendo del nivel en el que se localice el valor de salida del dato de prueba, se determinará la clase en la cual el punto será clasificado.

El segundo enfoque se refiere a cuando un modelo aMLP se comprime utilizando V-NN, donde cada MLP está especializada en una clase en particular, siendo posible identificar solamente una clase por cada una de la señales de salida. Por consiguiente, cada modelo V-NN se especializa en una clase y el análisis de la señal de salida determina si un patrón de prueba es parte de la clase o no. Durante la fase de entrenamiento, los datos se muestran en forma ordenada a cada modelo, presentando en primer lugar aquellos patrones que pertenecen a la clase. Por ejemplo, si el arreglo tiene tres modelos, el primero se entrena con datos en los que están activos los patrones correspondientes a la primer clase (aquellos que tienen un 1 como valor deseado) y los demás patrones, que no representan a la primer clase, no están activos (aquellos que tienen un 0 como valor deseado). Análogamente se presentan a sus correspondiente modelos los datos de entrenamiento para aprender la segunda y tercer clase.

La Figura 3.4 muestra el análisis de la señal de salida que debe realizarse para determinar los umbrales máximo y mínimo para cada clase en un arreglo. Es posible ver las señales de salida correspondientes a los modelos MLP (linea completa) y V-NN (línea punteada) para los datos de entrenamiento. Los umbrales inferiores y superiores (líneas discontinuas) de la clase se miden para poder determinar el cambio de la señal de salida para el modelo V-NN, considerando los valores obtenidos para el conjunto de entrenamiento. Estos cambios (umbrales) determinarán la pertenencia (o no) a la clase de un nuevo punto de datos que se recibe para clasificación. La membresía de un nuevo patrón (punto de prueba) a la clase se identifica analizando si la salida asociada a este dato tiene un valor dentro de los umbrales mínimo y máximo de la clase.

Como se está analizando la salida de un modelo aMLP, deben considerarse a diferentes señales, cada una correspondiente a una clase, para identificar cuál es la clase que se activa como resultado. Si más de un modelo fue activado por un dato de prueba, se aplica un criterio de *máximo*, que se explica a continuación.

Supongamos un modelo aMLP formado por C redes tales como las que se muestran en la Figura 3.2. Cada salida de la red toma un valor de 1 si la clase es identificada, o 0 de otro modo. La primer capa de cada MLP en el aMLP es un conjunto de N_I neuronas de entrada, y hay N_H neuronas ocultas. Cuando un

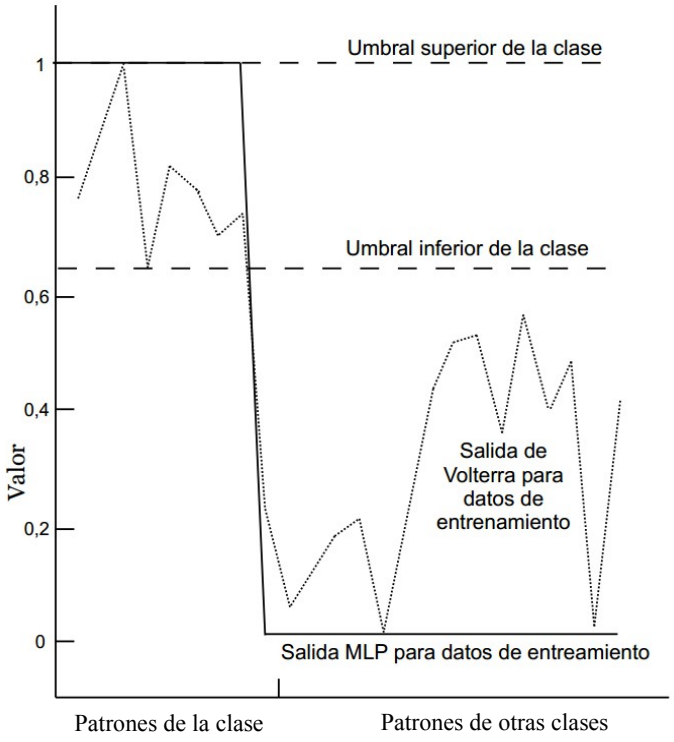

Figura 3.4: Análisis de la señal de salida de un clasificador comprimido. Líneas completas y punteadas: señal de salida correspondiente a los modelos MLP y V-NN, respectivamente, para datos de entrenamiento. Líneas discontínuas: umbrales de la clase.

patrón de datos debe ser clasificado, se presenta como entrada al clasificador, esto es, se lo presenta a todas las C redes definidas en el modelo aMLP, y la salida máxima obtenida entre todas las redes es asignada como etiqueta de la clase. Si se presenta al modelo un patrón de la c-ésima clase, se espera un valor (cercano a) 1 en la c-ésima red.

3.4. Medida para la selección de un modelo.

Las diferentes soluciones a un problema pueden evaluarse calculando las medidas de desempeño introducidas en el Capítulo 2. Cada salida del modelo V-NN tiene asociado un valor en cuanto a la tasa de reconocimiento y a la medida de ahorro de espacio. Sin embargo, a veces estos valores pueden tener significados completamente opuestos, surgiendo asi el problema de cómo seleccionar el mejor modelo. Por ejemplo, una solución puede tener un alto desempeño en cuanto a RR pero una tasa SS muy baja, o viceversa.

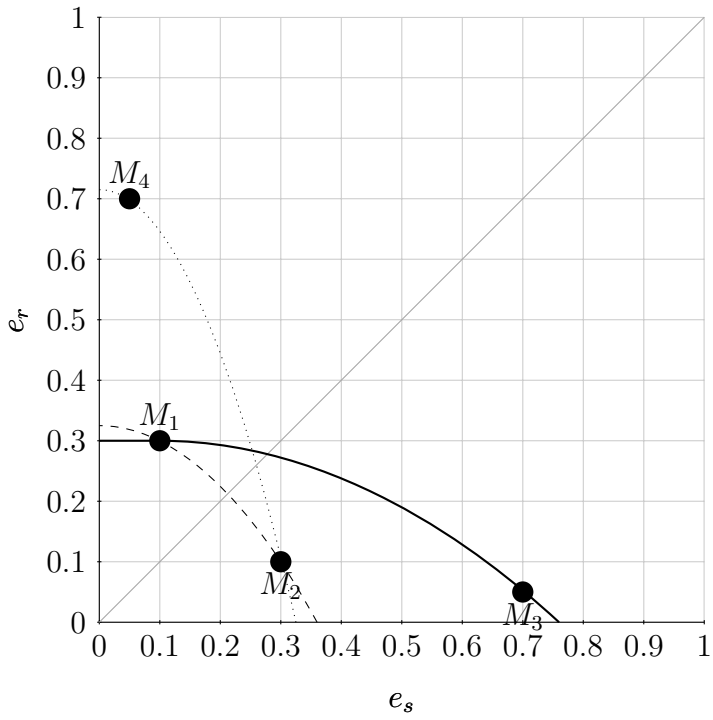

Figura 3.5: Selección de modelo mediante el análisis de las posibles soluciones para compresión en el espacio de los errores.

Es posible considerar el error para ambas medidas como un punto en una gráfica de ejes de coordenadas, donde el eje de las abcisas representa a $e_s = 1-SS$, y el eje de las ordenadas está asociado a $e_r = 1 - RR$. Por lo tanto, un punto (modelo) puede definirse como un par [e_s,e_r], siendo [0,0] el óptimo. La Figura 3.5 muestra ejemplos de modelos y sus errores asociados en este nuevo espacio. Supongamos un modelo 1 que posee SS=0,9 y RR= $0,7$, y un error al óptimo representado en el punto $M_1 = [0,1; 0,3]$. Para el modelo 2, donde SS=0,7 y RR=0,9, el punto es $M_2 = [0,3; 0,1]$. El modelo 3, en el que SS=0,3 y RR=0,95, es $M_3 = [0,7; 0,05]$; y un modelo 4, con SS=0,95 y RR=0,3, se representa con el punto $M_4 = [0,05; 0,7]$.

Considerando estos puntos, se puede plantear una solución de compromiso como la mínima distancia Euclídea de un punto $[e_s, e_r]$ al óptimo [0,0]. Sin embargo, cuando se calculan estas distancias, es posible tener casos con el mismo resultado pero valores completamente opuestos. Por ejemplo, los puntos M_1 y M_2 están equidistantes de la recta diagonal que va de (0;0) a (1;1) del eje de coordenadas, y tienen la misma distancia Euclídea al óptimo; similarmente a cualquier otro modelo que puede estar localizado en la parábola de línea discontínua gra-

ficada en la Figura 3.5. Sin embargo, M_2 es mejor clasificador que M_1, mientras que M_1 comprime más que el primero. Por lo tanto, es necesario discriminar cuál de los dos puntos, que representan modelos, es la mejor solución para el problema bajo estudio.

Es razonable pensar que para algunos problemas particulares de clasificación, por ejemplo reconocimiento de rostros, es más importante tener un mejor valor de tasa de reconocimiento antes que una alta tasa de ahorro de espacio. Por esto, es posible inferir que la solución de compromiso para este tipo de problemas debería estar siempre por encima de la recta diagonal, ya que es mejor tener una alta tasa de reconocimiento antes que una alta tasa de ahorro de espacio. Sin embargo, para otro tipo de aplicaciones que, por ejemplo, deban correr en un equipo celular, puede sugerirse una mejor compresión del modelo antes que un excelente clasificador; en este caso, SS sería más importante que RR, y la solución de compromiso debería estar por debajo de la recta diagonal.

Para poder evaluar cuál es el modelo más adecuado como solución para un problema dado, se propone una nueva medida ∂ de distancia que representa un compromiso entre RR y SS como se muestra a continuación:

$$\partial = \sqrt{(\gamma e_r)^2 + ((1-\gamma)e_s)^2}, \tag{3.17}$$

donde γ es un parámetro de pesado. Precisamente, para ser capaz de realizar la discriminación indicada anteriormente, se propone usar un parámetro de regularización γ. Para poder considerar a γ como un peso que permita seleccionar entre dos modelos, su valor puede modificarse de acuerdo a si se necesita un mejor compresor o un mejor clasificador, con $\gamma \, \epsilon \, [0; \ldots; 1]$.

Cuando $\gamma = 0{,}5$, tanto e_r como e_s son considerados con la misma importancia; la preferencia por RR es enfatizada con un $\gamma > 0,5$ (por ejemplo, la línea gruesa en la Figura 3.5), mientras la preferencia por SS se obtiene eligiendo un $\gamma < 0{,}5$ (por ejemplo, línea punteada en la Figura 3.5). Aplicando (3.17) a cada modelo presentado en la Figura 3.5 y considerando RR como una preferencia sobre SS con, por ejemplo $\gamma = 0{,}8$, se obtendrían las siguientes distancias al óptimo: $\partial_{M_1} = 0{,}241$, $\partial_{M_2} = 0{,}1$, $\partial_{M_3} = 0{,}146$, y $\partial_{M_4} = 0{,}56$. De este modo es posible concluir, sin ninguna duda, que la solución de mejor compromiso es M_2 para el ejemplo presentado anteriormente, ya que tiene la mínima distancia al óptimo, de acuerdo al cálculo de la nueva distancia propuesta.

3.5. Comprimiendo un clasificador

La propuesta realizada en este capítulo (Sección 3.3) para la compresión de un modelo neuronal, llamada Volterra-NN, es probada en esta sección con el problema de la base de datos Iris. Se muestra que varias topologías MLP pueden comprimirse adecuadamente en los pesos de Volterra de primer, segundo y tercer orden. Los resultados obtenidos ponen de manifiesto que estos pesos de Volterra pueden ser utilizados para construir un modelo de Volterra que necesita menos cantidad de parámetros que el modelo MLP original y que, sin embargo, posee la misma precisión que el modelo neuronal entrenado.

3.5.1. Datos y experimentos

En una prueba inicial del enfoque propuesto para compresión, se ha utilizado el conjunto de datos Iris[1] . Ésta es quizás la base de datos más conocida y utilizada en la literatura de reconocimiento de patrones (Duda y Hart, 2003). El conjunto de datos contiene 3 clases de 50 instancias cada una referidas a un tipo de especie norteamericana de la planta Iris (*Iris setosa*, *Iris versicolor* e *Iris virginica*). La primera de las clases es linealmente separable de las otras dos; la Iris versicolor no es linealmente separable de la Iris virginica. Cada patrón posee 4 atributos numéricos para cada flor: largo y ancho del sépalo y del pétalo.

En los experimentos, se ha utilizado validación cruzada con k particiones (Haykin, 2007) con el 80 % de cada clase para entrenar y el 20 % para testear. El dataset completo ha sido dividido aleatoriamente en $k = 5$ subconjuntos de datos de igual tamaño, con conjuntos de prueba disjuntos para cada k, repitiendo los experimentos tres veces en cada partición. La precisión global de cada modelo se ha calculado promediando las medidas de precisión obtenidas para los cinco conjuntos de datos de prueba.

Varias topologías MLP han sido evaluadas, de las cuales se han extraído luego del entrenamiento los correspondientes pesos de Volterra. Cada uno de los modelos MLP probados poseen cuatro variables de entrada, correspondientes a los atributos de las clases; y una neurona de salida indicando la etiqueta de la clase. Para la capa oculta han sido utilizados diferentes números de neuronas (4, 8

[1] http://archive.ics.uci.edu/ml/datasets/Iris

Tabla 3.2: Tasas de reconocimiento por clase (RR_i), para $i = 1$, 2 y 3, para los tres clasificadores MLP y sus correspondientes salidas V-NN de primer, segundo y tercer orden.

	$MLP_{4,4,1}$			$MLP_{4,8,1}$			$MLP_{4,12,1}$		
	RR_1	RR_2	RR_3	RR_1	RR_2	RR_3	RR_1	RR_2	RR_3
MLP	100,00 %	96,00 %	98,00 %	100,00 %	96,00 %	95,33 %	100,00 %	97,33 %	96,00 %
$s^{(1)}$	100,00 %	94,00 %	88,00 %	80,00 %	66,00 %	76,00 %	50,00 %	36,00 %	52,67 %
$s^{(2)}$	100,00 %	94,00 %	88,67 %	94,67 %	88,67 %	95,33 %	66,67 %	46,00 %	67,33 %
$s^{(3)}$	100,00 %	94,67 %	92,67 %	99,33 %	94,67 %	96,00 %	99,33 %	90,00 %	93,33 %

y 12). Para todas las clases, las unidades ocultas poseen una función de activación sigmoidea. Los valores iniciales de los pesos y umbrales para cada modelo han sido determinados utilizando una función aleatoria que provee un valor escalar extraído de una distribución uniforme en el intervalo unitario [0;1].

En cada experimento y repetición, los modelos MLP con menos del 90 % de tasa de clasificación para cada clase no han sido considerados para el análisis y extracción de los pesos de Volterra. Se ha impuesto esta restricción al desempeño del clasificador para poder medir precisamente cualquier pérdida de precisión originada por el método de compresión Volterra-NN propuesto.

3.5.2. Resultados y discusión

La Tabla 3.2 presenta los resultados obtenidos para la tasa de reconocimiento (RR_i) del modelo para cada clase $i = 1, 2, 3$ sobre el problema del conjunto de datos Iris. De cada modelo MLP_{N_I,N_H,N_O} (N_I, N_H y N_O representan la cantidad de neuronas en las capas de entrada, oculta y de salida, respectivamente) considerado en este estudio ($MLP_{4,4,1}, MLP_{4,8,1}$ y $MLP_{4,12,1}$), se han extraído sus correspondientes pesos de Volterra de orden cero, y de primer, segundo y tercer orden de acuerdo al Algoritmo 1, y sus correspondientes salidas de Volterra-NN $s^{(1)}$, $s^{(2)}$ y $s^{(3)}$ utilizando el Algoritmo 2.

En la tabla es posible observar que los tres modelos MLP muestran similares tasas de clasificación altas, de entre 95 % y 100 % para cada clase, con un valor promedio global para todas las clases de aproximadamente 98 %. Cuando un clasificador MLP tiene 4 neuronas ocultas ($MLP_{4,4,1}$), los valores de RR relacionados a la clase 1 son del 100 % para todas las salidas V-NN $s^{(1)}$, $s^{(2)}$ y $s^{(3)}$; para la segunda clase, estos valores varían entre 94 % y 96 %; y para la tercer

clase están entre 88 % y aproximadamente 93 %. Como ser verá en la Tabla 3.3, los valores promedios para estas salidas Volterra-NN varían entre 94 % y 95 %. Esto significa que, para estas arquitecturas MLP, puede mantenerse un desempeño clasificador promedio alto si se lo reemplaza con una salida V-NN de orden 1, 2 o 3. Puede observarse que una peor tasa RR es obtenida para este problema de clasificación si se utiliza $s^{(1)}$ en lugar de la MLP original, especialmente para la clase 3. Este valor bajo de RR es, aun así, de casi 90 %. Además, como se puede observar en la Tabla 3.3, el modelo V-NN $s^{(1)}$ requiere un número de parámetros significantemente más bajo que el clasificador original $MLP_{4,4,1}$ (una relación de 1:5). Otro hecho importante para tener en cuenta es que a pesar de que los valores RR para $s^{(3)}$ son los más altos, no hay compresión en este caso debido a que la cantidad de parámetros necesarios para construir este nuevo modelo es mayor que los requeridos por la MLP original.

Con respecto al clasificador $MLP_{4,8,1}$ el modelo Volterra-NN $s^{(1)}$ tiene una tasa de clasificación baja en comparación con la arquitectura neuronal original considerando todas las clases. Se puede notar una clara tendencia con respecto al hecho de que cuantos más pesos de Volterra se incluyen en la salida V-NN (el orden de la salida se incrementa), la tasa de reconocimiento mejora, casi alcanzando la tasa del clasificador original. Esto se logra, sin embargo, pagando el costo de incluir más parámetros en la salida y, por lo tanto, obteniendo menor compresión. Esta tendencia puede verse también en el modelo de 4 neuronas ocultas, a pesar de las pocas diferencias entre los valores RR para las salidas V-NN.

Es posible obtener conclusiones similares para el clasificador $MLP_{4,12,1}$. Sin embargo, para este modelo MLP y sus correspondientes resultados de V-NN, los valores de RR obtenidos son muy bajos para la salida V-NN de bajo orden. Esto significa que esta topología particular de clasificador no puede ser eficientemente reproducida por salidas de V-NN de bajo orden, y por lo tanto, utilizando un número bajo de parámetros. A pesar de este hecho, se puede afirmar que, en general, cuanto más parámetros se utilizan para construir las salidas V-NN, las tasas de reconocimiento se vuelven más altas, llegando incluso a valores muy cercanos al mismo desempeño del modelo MLP original.

Globalmente, si se considera que las tres topologías $MLP_{4,N_H,1}$ ($N_H = 4, 8, 12$) son propuestas como soluciones al mismo problema, en realidad no es ne-

Tabla 3.3: Comparación de las tasas de reconocimiento global (RR) y ahorro de espacio (SS) para tres clasificadores MLP y sus correspondientes salidas V-NN de primer, segundo y tercer orden.

		$MLP_{4,4,1}$		$MLP_{4,8,1}$		$MLP_{4,12,1}$	
$P_{MLP} \rightarrow$		25		49		73	
$RR \rightarrow$		98,00 %		97,11 %		97,78 %	
	P_{S_i}	RR	SS	RR	SS	RR	SS
$s^{(1)}$	5	94,00 %	80,00 %	74,00 %	89,90 %	46,22 %	93,15 %
$s^{(2)}$	15	94,22 %	40,00 %	92,89 %	69,39 %	60,00 %	79,45 %
$s^{(3)}$	31	95,11 %	−24,00 %	96,67 %	36,73 %	94,22 %	57,53 %

cesario considerar la topología $MLP_{4,12,1}$ porque la mejor solución está dada por $MLP_{4,4,1}$. Esto significa que hay un número de pesos innecesarios en $MLP_{4,12,1}$ que posiblemente contribuyen al sobreentrenamiento del modelo, causando un declive en la RR sobre el conjunto de datos de prueba. Estos pesos no pueden comprimirse eficientemente con el modelo Volterra-NN, arrojando RR bajas para las soluciones de este modelo.

La Tabla 3.3 muestra las tasas de ahorro de espacio (SS) para los modelos discutidos en la Tabla 3.2, y los valores medios de la tasa de reconocimiento (RR) para estos modelos. En primer lugar, se muestra en la primer fila el número de parámetros necesarios para cada arquitectura del clasificador neuronal (P_{MLP}) involucrados en este estudio, mientras que el número de pesos de Volterra necesarios para cada salida i de Volterra-NN P_{S_i} se muestra en la primer columna. Los valores que completan las columnas RR y SS son la tasa de reconocimiento y la capacidad de ahorro de espacio para cada combinación entre un clasificador MLP y su correspondiente salida Volterra-NN.

Como se ha indicado en el marco teórico, en la ecuación (2.33), el número de parámetros necesarios para cada clasificador neuronal depende no sólo de N_I sino también del número de neuronas en la capa oculta (N_H). El número de parámetros para el modelo $MLP_{4,4,1}$ es de 25; para $MLP_{4,8,1}$ es de 49; y el modelo $MLP_{4,12,1}$ requiere de 73 parámetros. En el caso de salidas de V-NN de diferente orden, el número de parámetros solamente depende del número de entradas. El número de parámetros para $s^{(1)}$ es 5 según la ecuación (3.12). Aplicando las ecuaciones (3.13) y (3.14) el número de parámetros necesarios para las salidas de Volterra-NN $s^{(2)}$ y $s^{(3)}$ es de 15 y 31, respectivamente.

Es posible ver que, cuando se usa el modelo Volterra-NN de primer orden $s^{(1)}$

en lugar del clasificador $MLP_{4,4,1}$, se obtiene una tasa de reconocimiento global del 94 % para el problema del Iris, y se alcanza una compresión o tasa de ahorro de espacio del 80 %. La mitad de esta tasa de ahorro de espacio se obtiene si se utiliza el modelo $s^{(2)}$, y utilizando $s^{(3)}$ no hay compresión. Esta ausencia de compresión se debe al hecho de que el número de parámetros necesarios para $s^{(3)}$ es más alto que el número de parámetros necesarios para el modelo $MLP_{4,4,1}$. Para el modelo $MLP_{4,8,1}$ se alcanza, mediante las salidas de Volterra-NN, una compresión más alta debido a la cantidad de parámetros asociados con este modelo; en este caso, el valor SS alcanza un máximo cercano al 90 % cuando se considera el número más pequeño de parámetros (una salida V-NN de primer orden). Un caso similar se presenta en el modelo $MLP_{4,12,1}$, donde se verifica esta tendencia.

De la Tabla 3.3 es posible concluir que se puede lograr comprimir bien un clasificador MLP utilizando un modelo V-NN; y esta compresión puede ser, en algunos casos, aún mayor al 90 % de los parámetros. En efecto, la arquitectura del modelo MLP puede ser más y más compleja, puede tener muchas más unidades ocultas, pero el número de pesos necesarios para construir cada salida V-NN sigue siendo el mismo mientras que el número de variables de entrada del problema sean las mismas, y esto se refleja, sin dudas, en valores aún más altos de las tasas de ahorro de espacio.

Sin embargo, en algunos casos, estas altas tasas de compresión tienen que pagarse con tasas de clasificación más bajas. Analizando esta tabla, puede notarse la influencia directa de la tasa de compresión sobre la tarea de clasificación. Para el modelo $MLP_{4,4,1}$, en esta experiencia, el mejor valor de RR se relaciona con el modelo V-NN $s^{(3)}$, alcanzando un 95,11 %. Pero, como se explicó anteriormente, en este caso no existe compresión. Por lo tanto, a pesar de que $s^{(2)}$ pueda considerarse como la mejor opción con su correspondiente SS del 40 %, es mejor seleccionar el modelo $s^{(1)}$ que posee casi el mismo valor RR pero logra un valor SS más alto: 80 %. Considerando el caso del modelo $MLP_{4,8,1}$, el mejor valor RR está asociado con el modelo $s^{(3)}$, el que tiene solamente una SS de 36,73 %. Si se requiere una mejor compresión para esta topología, es posible elegir el modelo $s^{(2)}$ que tiene una SS de 69,39 % y mantiene un valor alto de RR de 92,89 %.

Para el modelo $MLP_{4,12,1}$, el mejor valor de RR está asociado con la salida $s^{(3)}$ y es capaz de alcanzar un ahorro de espacio de $57,53\,\%$. Éste es un resultado muy interesante, porque con aproximadamente la mitad (31) de la cantidad de

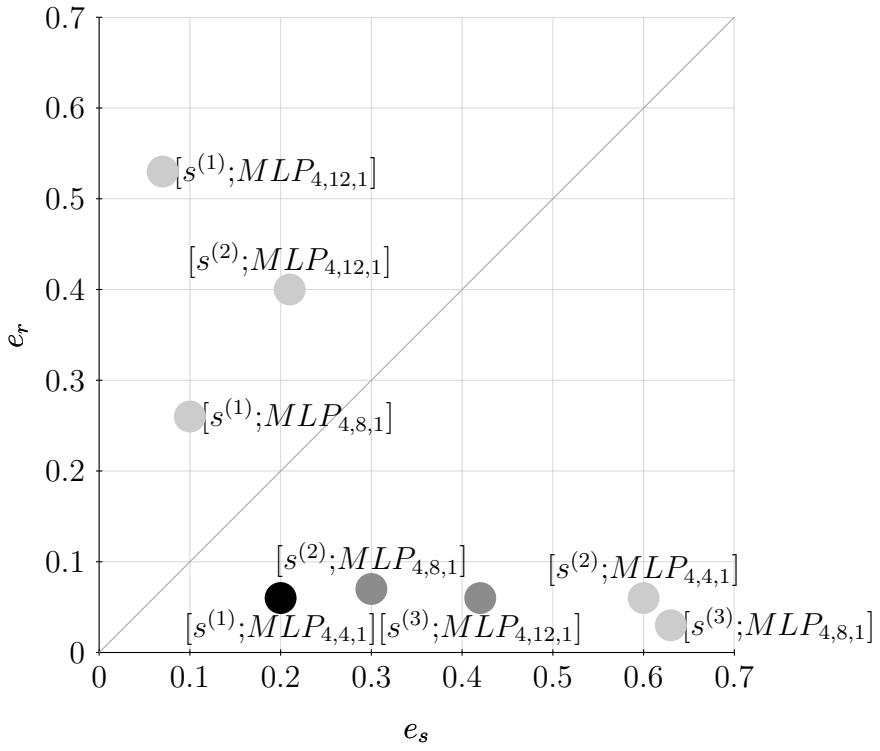

Figura 3.6: Selección del modelo perteneciente a la mejor solución de compromiso para los resultados observados en la Tabla 3.3.

parámetros que requiere el clasificador MLP original (73), la capacidad de clasificación del modelo $s^{(3)}$ es alta (94,22 %) y cercana al modelo MLP. Este caso particular puede considerarse como uno de los mejores resultados globales obtenidos en este caso de estudio, donde el clasificador $MLP_{4,12,1}$ puede comprimirse casi un 60 % utilizando el correspondiente modelo V-NN y su salida $s^{(3)}$ sin una pérdida significativa de la capacidad de reconocimiento.

Globalmente, la solución de mejor compromiso se obtiene utilizando el modelo $s^{(1)}$ cuando la topología del modelo MLP es $MLP_{4,4,1}$, el que tiene una tasa de reconocimiento casi similar al modelo $s^{(3)}$ en el caso de la topología $MLP_{4,12,1}$ (94 %), pero con una tasa de ahorro de espacio del 80 %. Esto puede ser visto también comparando estos dos casos en términos de número absoluto de parámetros. Es decir, mientras que para el modelo $s^{(1)}$ solamente se requieren 5 parámetros, en el caso del $MLP_{4,4,1}$, se necesitan 31 parámetros para el modelo $s^{(3)}$ cuando se considera la topología $MLP_{4,12,1}$.

Todo el análisis realizado en los párrafos anteriores puede verse mucho más fácil y rápidamente a través de la medida para la selección de modelos propuesta

Tabla 3.4: Selección de las mejores soluciones de compromiso ∂, con $\gamma = 0{,}6$, para las salidas del modelo Volterra-NN mostradas en la Tabla 3.3.

	$MLP_{4,4,1}$	$MLP_{4,8,1}$	$MLP_{4,12,1}$
$s^{(1)}$	**0,3406**	0,4432	0,5917
$s^{(2)}$	0,5241	**0,4063**	0,5676
$s^{(3)}$	0,7248	0,5226	**0,4523**

en la sección 3.4. En la Tabla 3.4 se muestra el cálculo de la medida ∂ para la selección de los modelos, considerando un $\gamma = 0{,}6$, es decir, dándole levemente más importancia a RR que a SS. Es posible observar, con los valores resaltados en negrita, que las mejores soluciones coinciden con las indicadas anteriormente. Además, las soluciones antes mencionadas pueden verse gráficamente en la Figura 3.6. Cada uno de los círculos representa una solución basada en la compresión de un modelo clasificador MLP_{N_I,N_H,N_O} mediante $s^{(i)}$, donde i es el orden de la salida V-NN. Los círculos grises oscuros y negro corresponden a las mejores soluciones de compromiso para este problema (siendo el negro el mejor de todos), cuyo análisis hecho anteriormente desde las Tablas 3.3 y 3.4 puede corroborarse gráficamente ya que son los tres puntos más cercanos al óptimo (el origen del sistema cartesiano). La solución de mejor compromiso global antes mencionada se evidencia muy rápidamente en la gráfica y concuerda con el análisis anterior, ya que el punto más cercano al óptimo es el que está asociado a utilizar la salida $s^{(1)}$ para comprimir el modelo $MLP_{4,4,1}$.

3.6. Comprimiendo un arreglo de clasificadores

El objetivo de esta sección es extender la evaluación del método de compresión de un modelo clasificador, basado en la aplicación del método V-NN a un arreglo de MLP. Para esto, se mostrará cómo un modelo *a*MLP que ha aprendido un problema de clasificación con cierta (alta) precisión, puede comprimirse en una representación más compacta utilizando un arreglo de modelos Volterra-NN. Esta representación involucra menos parámetros, manteniendo sin embargo una alta precisión en cuanto a la tasa de reconocimiento. En este caso de estudio se utilizaron dos bases de datos diferentes para mostrar la efectividad del método

Tabla 3.5: Conjunto de datos utilizados para la evaluación extendida del método Volterra-NN con arreglos de clasificadores neuronales.

Conjunto de datos	clases	variables	entrenamiento	prueba
Pendigits	10	16	8400	2100
Letters	26	16	15184	3796

propuesto en problemas de clasificación de diferente nivel de complejidad, número de clases y tipos de aplicación.

3.6.1. Datos y experimentos

En este caso de estudio se han utilizado dos bases de datos independientes. La primera de ellas fue construida para el reconocimiento de letras[2] (nombrada aquí como Letters, de su nombre en inglés Letter Recognition). La segunda es utilizada para el reconocimiento de dígitos escritos a mano[3] (nombrada como Pendigits, de su nombre en inglés Pen-Based Recognition of Handwritten Digits).

La Tabla 3.5 presenta las características de las dos bases de datos mencionadas anteriormente. El nombre de la base de datos se muestra en la primera columna, mientras que en la segunda y tercer columna se presentan el número de clases y el número de variables de la bases de datos, respectivamente. El número de datos en el conjunto de entrenamiento se muestran en la cuarta columna, como así también se presenta el número de patrones en el conjunto de prueba en la quinta columna.

En el problema de reconocimiento de letras, los conjuntos de entrenamiento y prueba tienen 16 variables que representan la información de los atributos de una determinada letra capital. Por lo tanto, cada modelo MLP asociado con cada clase consiste de 16 neuronas en la capa de entrada y solamente 1 neurona de salida (letra capital). Con respecto a la cantidad de neuronas en la capa oculta, se adopta la heurística simple en la que el número de neuronas en la capa oculta N_H es el mismo, el doble y el triple del número de entradas N_I. Por lo que N_H será de 16, 32 y 48. En el caso de los datos de la base de datos de los dígitos escritos a mano, el conjunto de datos consiste de 16 atributos que representan cada clase.

[2] http://archive.ics.uci.edu/ml/datasets/Letter+Recognition
[3] http://archive.ics.uci.edu/ml/datasets/Pen-Based+Recognition+of+Handwritten+Digits

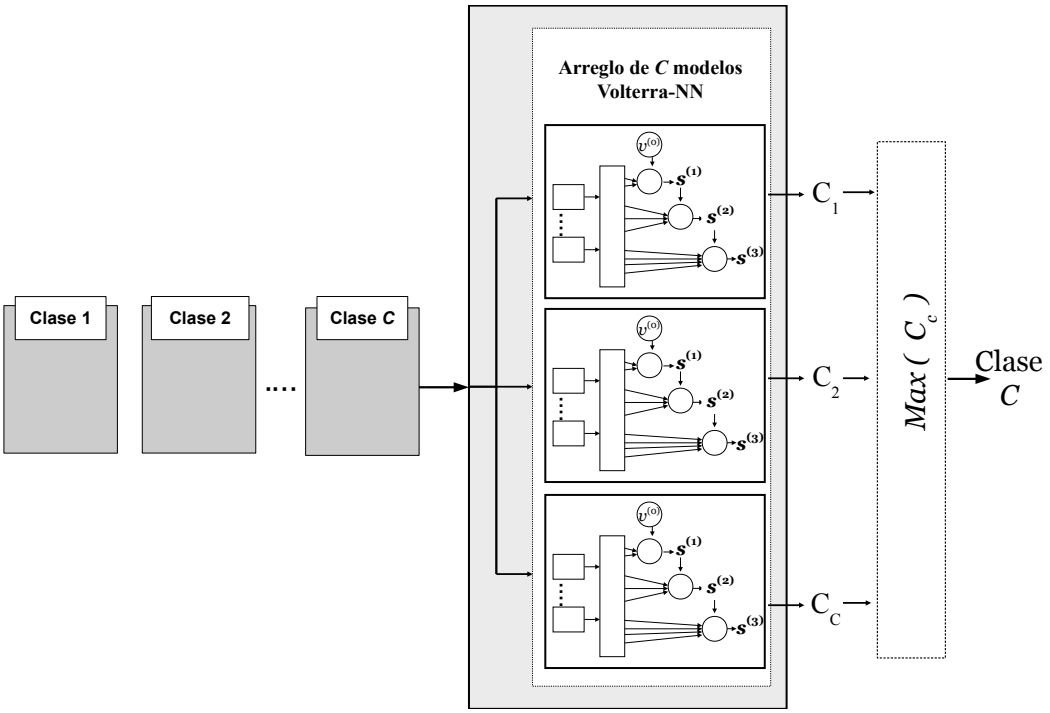

Figura 3.7: Arreglo de V-NN para problemas de clasificación.

Por lo tanto, cada modelo MLP tiene 16 neuronas de entrada y solamente 1 salida. El número de neuronas ocultas es de 16, 32 y 48.

Como en los casos de estudio anteriores, se ha utilizado el procedimiento de validación cruzada con k particiones (Haykin, 2007), utilizando la configuración estándar de dividir los patrones de datos disponibles en un 80 % de cada clase para el entrenamiento y un 20 % para prueba. El conjunto de datos completo ha sido dividido en $k = 3$ subconjuntos de igual tamaño, repitiendo los experimentos tres veces en cada partición. La estimación de la precisión global de un modelo, se ha calculado promediando las medidas de precisión sobre el conjunto de datos de prueba, en cada uno de los cuales se usan diferentes patrones (conjuntos disjuntos). En cada experimento y repetición, los modelos MLP que poseían menos del 95 % de tasa de clasificación para cada clase del conjunto de entrenamiento no han sido considerados en el estudio, por la razón mencionada en la sección 3.5.1.

La Figura 3.7 muestra la topología de la red neuronal utilizada en este estudio, un arreglo de modelos MLP, cada uno asociado con una clase a reconocer. Varias topologías MLP fueron evaluadas, de las cuales se han extraído los correspondientes pesos de Volterra luego del entrenamiento, con el objetivo de construir

los modelos V-NN. Para cada señal de entrada al arreglo, puede obtenerse una salida V-NN de cierto orden. Por ejemplo, para obtener una salida de primer orden, se selecciona el máximo $s^{(1)}$ del arreglo. Los parámetros del modelo (pesos y umbrales) son inicializados aleatoriamente con valores uniformemente distribuídos entre 0 y 1. Las neuronas en las capas oculta tienen una función de activación sigmoidea. Todas las MLPs son entrenadas con el algoritmo Levenberg-Marquardt (Marquardt, 1963).

3.6.2. Resultados y discusión

En esta subsección se muestran los resultados experimentales obtenidos sobre los dos problemas de clasificación presentados anteriormente. Por simplicidad en el análisis de los resultados, en particular con respecto al desempeño de las capacidades de compresión de V-NN, las tablas de resultados solamente muestran el RR global para cada base de datos (el valor RR promedio sobre todas las clases) y la tasa de ahorro de espacio para los modelos discutidos aquí.

A partir de cada modelo arreglo de MLPs considerado, con N_I neuronas de entrada, N_H neuronas ocultas, y N_O neuronas de salida ($aMLP_{N_I,N_H,N_O}$), han sido extraídos sus correspondientes pesos de Volterra de primer, segundo y tercer orden de acuerdo al Algoritmo 1. Luego, han sido obtenidas sus correspondientes salidas $s^{(1)}$, $s^{(2)}$ y $s^{(3)}$ del arreglo de Volterra-NN utilizando el Algoritmo 2.

La Tabla 3.6 presenta los resultados de la RR global y de la SS, calculadas sobre el conjunto de datos de prueba de la base Pendigits; mientras que la Tabla 3.7 muestra resultados similares pero calculados sobre el conjunto de datos de prueba de la base Letters. Los valores que completan las columnas RR y SS son la tasa de reconocimiento promedio y las capacidades de ahorro de espacio, respectivamente, para cada combinación entre un clasificador aMLP y su correspondiente salida aV-NN para las 3 particiones utilizadas.

Ambas tablas presentan los resultados de manera similar a como se ha realizado en la sección anterior. La diferencia radica en que, en este caso, las topologías corresponden al tipo arreglo de clasificadores del tipo $aMLP_{N_I,N_H,N_O}$. Por lo tanto, es diferente el cálculo del número de parámetros necesarios para cada arquitectura de clasificador neuronal (P_{aMLP}), como así también del número de pesos de Volterra necesarios para cada modelo Volterra-NN ($P_{as^{(i)}}$).

Tabla 3.6: Comparación de las tasas de reconocimiento global (RR) y de ahorro de espacio (SS) para tres clasificadores aV-NN (a=10) y sus correspondientes salidas de primer $s^{(1)}$, segundo $s^{(2)}$ y tercer orden $s^{(3)}$, para la base de datos de dígitos manuales escritos con lapicera.

		$10MLP_{16,16,1}$		$10MLP_{16,32,1}$		$10MLP_{16,48,1}$	
$P_{aMLP_{N_I,N_H,N_O}} \rightarrow$		3040		6080		9120	
RR[%] $\rightarrow$		98,47		98,61		98,58	
*10*V-NN	$P_{as^{(i)}}$	RR[%]	SS[%]	RR[%]	SS[%]	RR[%]	SS[%]
*10*V-NN$_{s^{(1)}}$	160	84,20	94,74	83,92	97,37	82,46	98,25
*10*V-NN$_{s^{(2)}}$	1520	83,18	50,00	84,01	75,00	81,73	83,33
*10*V-NN$_{s^{(3)}}$	4080	84,14	−34,21	84,56	32,89	84,35	55,26

Tabla 3.7: Comparación de las tasas de reconocimiento global (RR) y de ahorro de espacio (SS) para tres clasificadores aV-NN (a=26) y sus correspondientes salidas de primer $s^{(1)}$, segundo $s^{(2)}$ y tercer orden $s^{(3)}$, para la base de datos de reconocimiento de letras.

		$26MLP_{16,16,1}$		$26MLP_{16,32,1}$		$26MLP_{16,48,1}$	
$P_{aMLP_{N_I,N_H,N_O}} \rightarrow$		7904		15808		23712	
RR[%] $\rightarrow$		97,41		97,70		97,67	
*26*V-NN	$P_{as^{(i)}}$	RR[%]	SS[%]	RR[%]	SS[%]	RR[%]	SS[%]
*26*V-NN$_{s^{(1)}}$	416	92,37	94,74	92,93	97,37	92,78	98,25
*26*V-NN$_{s^{(2)}}$	3952	92,81	50,00	93,17	75,00	93,25	83,33
*26*V-NN$_{s^{(3)}}$	10608	92,83	−34,21	93,24	32,89	92,70	55,26

Como se dijo en el marco teórico (sección 2.4), el número de parámetros requerido para un clasificador neuronal depende no solamente de N_I sino del número de neuronas en la capa oculta N_H, como así también del número de elementos en el arreglo. Por lo tanto, considerando la base de datos Pendigits, el número de parámetros para *10*MLP$_{16,16,1}$ es 3040, para *10*MLP$_{16,32,1}$ es de 6080, y el modelo *10*MLP$_{16,48,1}$ tiene un total de 9120 parámetros (ver Tabla 3.6). Con respecto a la base de datos de letras, el número de parámetros para *26*MLP$_{16,16,1}$ es 7904, para *26*MLP$_{16,32,1}$ es de 15808, y el modelo *26*MLP$_{16,48,1}$ tiene un total de 23712 parametros (ver Tabla 3.7).

En el caso de las salidas V-NN de diferente orden, el número de parámetros solamente depende en el número de entradas, como se ha mostrado en la subsección 3.3.1. Sin embargo, para un arreglo, este número debe ser multiplicado por el tamaño del arreglo. Por lo tanto, el número de parámetros para *10*V-NN$_{s^{(1)}}$ es 160 en el modelo de la base Pendigits, y el número de parámetros necesarios para *10*V-NN$_{s^{(2)}}$ y *10*V-NN$_{s^{(3)}}$ son 1520 y 4080, respectivamente. Con respecto a

la base Letters, el número de parámetros es 416, 3952 y 10608 para los modelos *26*V-NN$_{s^{(1)}}$, *26*V-NN$_{s^{(2)}}$ y *26*V-NN$_{s^{(3)}}$, respectivamente.

Enfocándose en la Tabla 3.6, es posible obervar que cuando se utiliza a *10*V-NN$_{s^{(1)}}$ en lugar del clasificador *10*MLP$_{16,16,1}$, se obtiene una tasa de reconocimiento global del 84,20 % y una tasa de ahorro de espacio del 94,74 %. Casi la mitad de esta tasa de ahorro de espacio se obtiene si se utiliza la salida *10*V-NN$_{s^{(2)}}$, y no hay ahorro de espacio cuando se utiliza *10*V-NN$_{s^{(3)}}$. Para el modelo *10*MLP$_{16,32,1}$, la compresión alcanzada por los modelos Volterra-NN es más alta debido a la cantidad de parámetros asociados a este modelo; en este caso, el valor SS alcanza un máximo de 97,37 % cuando se considera el menor número posible de parámetros (una salida Volterra-NN de primer orden). Un caso similar está relacionado con el modelo *10*MLP$_{16,48,1}$, en el que se verifica esta tendencia.

A partir de los resultados expuestos en la Tabla 3.7, puede obtenerse una tasa de reconocimiento global del 92,37 % y puede alcanzarse una tasa de ahorro de espacio de 94,74 % cuando se utiliza la salida *26*V-NN$_{s^{(1)}}$ en lugar del clasificador *26*MLP$_{16,16,1}$ en el caso de el reconocimiento de letras. Considerando el caso de *26*MLP$_{16,32,1}$, el mejor valor RR está asociado con la salida *26*V-NN$_{s^{(2)}}$, la que tiene solamente una SS de 75 %. Pero si se necesita una mejor compresión para esta topología, es posible elegir la salida *26*V-NN$_{s^{(1)}}$, la que tiene una SS del 97,37 % y preserva un valor RR muy alto, 92,93 %. El mejor valor RR para el modelo *26*MLP$_{16,48,1}$ está asociado con la salida del modelo Volterra-NN *26*V-NN$_{s^{(2)}}$ y es posible alcanzar para ésta una SS de 83,33 %. Pero, si se elige la salida *26*V-NN$_{s^{(1)}}$, el valor de RR apenas empeora unas pocas décimas, y se obtiene un valor muy alto de SS del 98,25 %. Este es un resultado muy interesante, porque con aproximadamente menos del 2 % de los parámetros del clasificador *a*MLP original, la capacidad de clasificación de *26*V-NN$_{s^{(1)}}$ es muy alta (92,78 %) y muy cercana a la del modelo *a*MLP. Este caso particular puede considerarse como el mejor resultado global obtenido en este caso de estudio, donde el clasificador *26*MLP$_{16,48,1}$ puede comprimirse en un 98 % utilizando la correspondiente salida *26*V-NN$_{s^{(1)}}$ sin pérdida significativa de capacidad de reconocimiento.

De las dos tablas analizadas anteriormente, es posible concluir que un arreglo de MLPs para clasificación puede comprimirse adecuadamente utilizando modelos de arreglo de Volterra-NN; y esta tasa de compresión puede ser, en algunos casos, aún mayor al 90 %. Más aún, en muchos casos, se alcanzan tasas de reconocimiento

Tabla 3.8: Selección de las mejores soluciones de compromiso ∂, con $\gamma = 0{,}6$, para las salidas del modelo aV-NN ($a = 10$) mostradas en la Tabla 3.6.

	$10MLP_{16,16,1}$	$10MLP_{16,32,1}$	$10MLP_{16,48,1}$
10V-NN$_{s^{(1)}}$	**0,3404**	**0,3271**	**0,3350**
10V-NN$_{s^{(2)}}$	0,5486	0,4427	0,4199
10V-NN$_{s^{(3)}}$	0,7950	0,6009	0,5224

Tabla 3.9: Selección de las mejores soluciones de compromiso ∂, con $\gamma = 0{,}6$, para las salidas del modelo aV-NN ($a = 26$) mostradas en la Tabla 3.7.

	$26MLP_{16,16,1}$	$26MLP_{16,32,1}$	$26MLP_{16,48,1}$
26V-NN$_{s^{(1)}}$	**0.2585**	**0.2301**	**0.2243**
26V-NN$_{s^{(2)}}$	0,4931	0,3755	0,3274
26V-NN$_{s^{(3)}}$	0,7615	0,5559	0,4720

altas.

Para la base de datos Pendigits se alcanzan valores altos de ahorro de espacio, pero pagando un costo de tasa de reconocimiento más baja. Por ejemplo, para los tres modelos que se muestran en la Tabla 3.6, los valores SS que están entre 95 % y 98 % aproximadamente, se relacionan a valores RR entre 82,46 % y 84,20 %. A pesar de estos resultados, es posible ver que los valores de RR son aún altos. Es importante destacar entonces que, en la mayoría de los casos estudiados, estas altas tasas de compresión no deben pagarse con tasas de clasificación muy bajas. Por ejemplo, para el modelo 26MLP$_{16,16,1}$ en la base Letters, el mejor valor RR está asociado con la salida 26V-NN$_{s^{(2)}}$, alcanzando un valor de 92,81 %, y teniendo una tasa de ahorro de espacio del 50 %. Pero, si se escoge el mejor resultado para SS (94,74 %), es posible obtener un valor de más del 92 % para RR (ver Tabla 3.7).

Es importante dejar en claro que los valores de SS para todos los casos en la Tabla 3.7 son los mismos. Como se indicó anteriormente, el valor de SS se obtiene a partir del número de parámetros de cada modelo V-NN, el que está basado en el número de parámetros de entrada para el modelo MLP. Como es posible ver en la Tabla 3.6, tanto para la base Pendigits como para Letters, el número de parámetros es 16. Por esto, independientemente del número de clases presentes en

ambas bases de datos, el valor de SS es exactamente el mismo cuando el número de parámetros en la entrada es el mismo.

Luego de analizar los resultados en este caso de estudio, es posible pensar que los mejores resultados globales se obtienen siempre en modelos con una cantidad alta, y posiblemente innecesaria, de neuronas ocultas. Pero, sin lugar a dudas, en los dos problemas de clasificación que se estudiaron en esta sección, si el modelo aMLP con el mismo número de neuronas ocultas que el de entrada, es reemplazado por la salida aV-NN$_{s^{(1)}}$, los valores de RR obtenidos son cercanos a los del modelo original, y el ahorro de espacio es definitivamente alto.

En las Tablas 3.8 y 3.9 se muestra el cálculo de la medida ∂ para la selección de los modelos, considerando un $\gamma = 0{,}6$ y los resultados expuestos en las Tablas 3.6 y 3.7 respectivamente. Es posible observar, con los valores resaltados en negrita, que las mejores soluciones coinciden con las indicadas en los párrafos anteriores.

3.7. Aplicación a un problema real de reconocimiento de rostros

Muchas áreas de aplicación aprovechan esta capacidad de los arreglos y los ensambles de clasificadores para lograr mejores resultados que los modelos por sí solos, en particular, para la tarea de Reconocimiento de Rostro (FR, de su traducción al inglés Face Recognition) (Zhang y Wangmeng, 2007). En esta sección se mostrará la aplicación del modelo aV-NN propuesto al reconocimiento de rostros, el cual involucra un problema real más complejo que los ya analizados, y un número mayor de clases.

3.7.1. Datos y experimentos

En un sistema completo de reconocimiento de rostros, la primera etapa consiste en la detección de la cara a través de alguna técnica de procesamiento de imágenes. En segundo lugar, se aplica un método de extracción de características para obtener la información útil de la cara. Puede realizarse una representación global a través del uso de una técnica conocida como *eigenfaces* (Turk y Pentland, 1991), mediante la aplicación del análisis de las componentes principales (PCA,

Figura 3.8: Ejemplos de imágenes de rostros de la base de datos de rostros AT&T Laboratories Cambridge ORL (Li y Jain, 2004).

de su traducción al inglés Principal Component Analysis) para la reducción de la dimensionalidad (Kirby y Sirovich, 1990), el cual es el método de extracción de características más ampliamente utilizado para FR (Li y Jain, 2004). Finalmente, esta información es utilizada en un clasificador para el reconocimiento de los rostros (Zhao y otros, 2003). Éste puede ser un método simple tal como la distancia Euclídea o el de los K vecinos más cercanos, o una solución basada en un arreglo o ensamble de clasificadores débiles, siendo el modelo MLP uno de los modelos más populares (Martinez y Kak, 2001) (Kong y otros, 2005).

Un modelo *a*MLP fue propuesto en (Capello y otros, 2009) para la tarea de clasificación en un sistema de FR, consistiendo en una MLP por cada sujeto (persona válida o autorizada), con una decisión final tomada en base a las salidas de la red del arreglo completo. Esta configuración ha permitido obtener significativas mejoras sobre la performance de una única MLP clásica. Sin embargo, el uso de esta nueva configuración implica una cantidad mayor de parámetros, y por lo tanto, un sistema FR más grande y más complejo. Debido al hecho de este tipo de modelos deberían ser capaces de correr en pequeños dispositivos de procesamiento tales como por ejemplo los teléfonos móviles, iPads y cámaras de seguridad, o ser transmitidos on-line, el modelo debe tener un tamaño apropiado o ser comprimido con el objeto de ajustarse a estos requerimientos.

Para demostrar la efectividad de la aplicación del modelo Volterra-NN a un arreglo de MLP, han sido utilizadas dos bases de datos diferentes de reconocimiento de rostros, evaluando el desempeño del compresor propuesto para un arreglo de clasificadores. Estas bases de datos proporcionan configuraciones experimentales típicas para el reconocimiento de rostros.

Los primeros experimentos fueron realizados utilizando la base de datos de

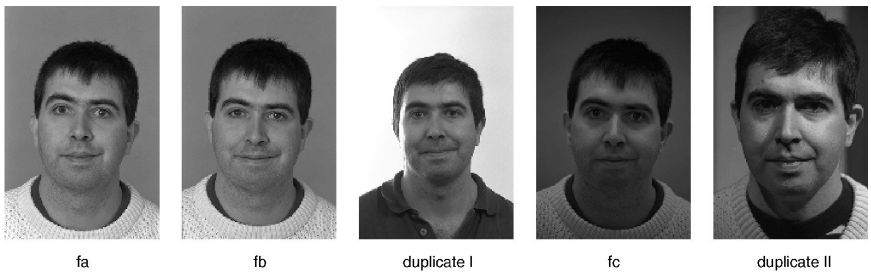

Figura 3.9: La base de datos FERET (sigla de su traducción al inglés Facial Recognition Technology) (Phillips y otros, 2000). Ejemplos de diferentes categorías de pruebas (imágenes).

rostros ORL de AT&T Laboratories Cambridge [4] que es ampliamente utilizada en la literatura de reconocimiento de rostro (Li y Jain, 2004). En esta base de datos, hay 10 imágenes diferentes de cada una de las 40 personas de diferente género, raza y edad (ver Figura 3.8). Para algunos sujetos, las imágenes han sido tomadas en diferentes instantes de tiempo, variando la iluminación, las expresiones faciales (ojos cerrados/abiertos, sonriendo/serios) y detalles faciales (con anteojos/sin anteojos). Todas las fotografías fueron tomadas contra un fondo negro homogéneo con los sujetos en una posición vertical y frontal, contemplando cierta tolerancia en cuanto a algunos movimientos de lado. La base de datos completa contiene 400 imágenes de rostros en escala de grises de un tamaño de 92 x 112 pixeles.

El entrenamiento de diferentes clasificadores para cada una de las clases, que en este caso de estudio representan a un sujeto, requiere datos de entrenamiento suficientes para cada clase. Sin embargo, en tareas de reconocimiento de rostros es común tener un número pequeño de fotografías por persona. De hecho, como en este caso se utilizan diferentes particiones de datos para entrenar y validar al clasificador, solamente dos fotos por sujeto están disponibles para prueba. Por lo tanto, se ha realizado un agregado de ruido al conjunto de datos de prueba con el objetivo de ampliarlo y así obtener más ejemplos de prueba para evaluar el desempeño del modelo V-NN. El ruido utilizado fue *Gaussiano* con una media de 0 y una varianza entre 0,01 y 0,1. Luego de este procedimiento, se obtuvieron un total de 66 patrones de datos por clase para cada conjunto de prueba.

Por su parte, la base de datos FERET (Phillips y otros, 2000) fue desarrollada desde 1993 hasta 1997[5] , patrocinada por el Programa de Desarrollo

[4] www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html
[5] www.itl.nist.gov/iad/humanid/feret

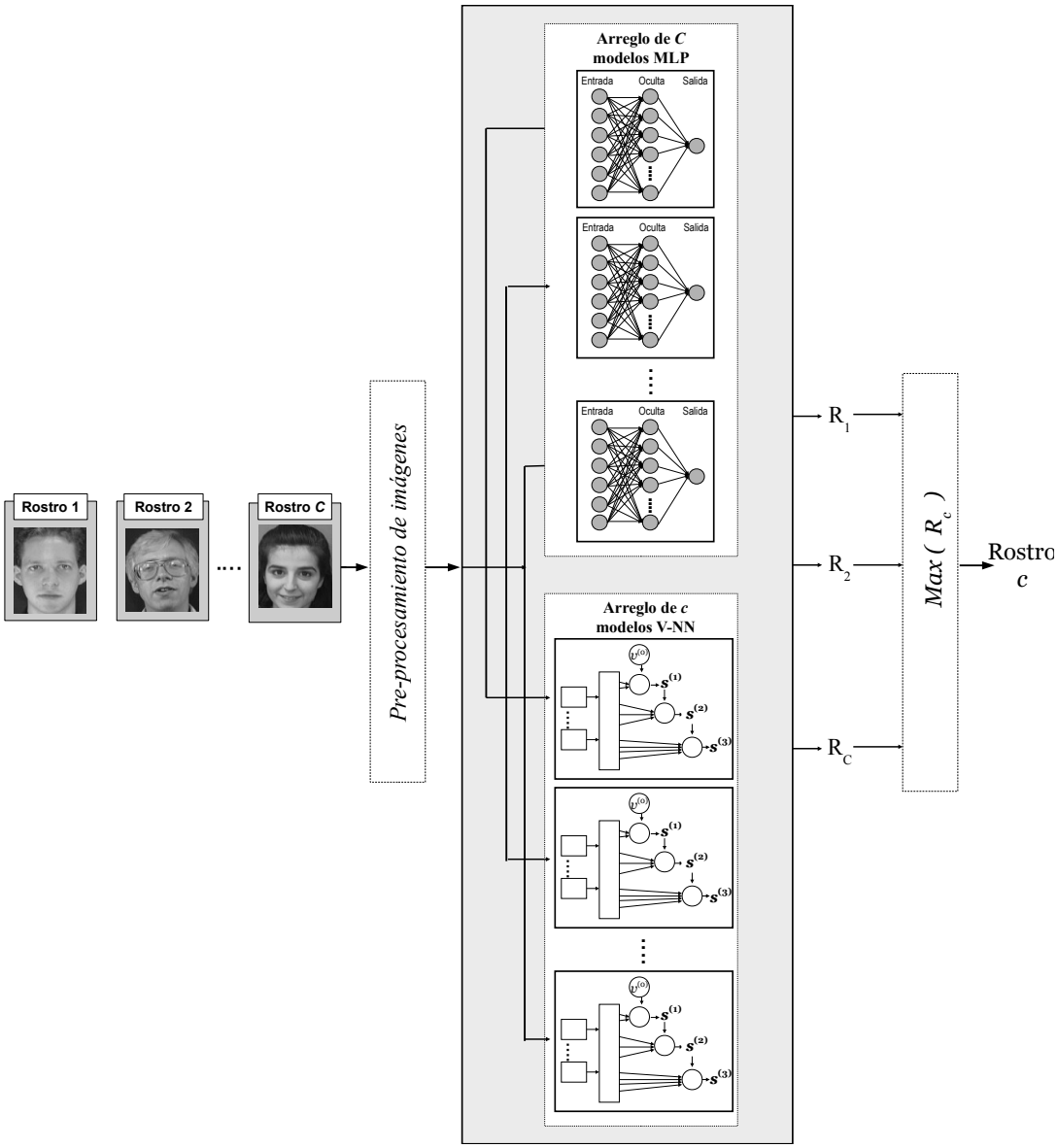

Figura 3.10: Modelo *a*MLP (parte superior del clasificador) para un problema de reconocimiento de rostro, que puede comprimirse en un arrego de modelos V-NN (parte inferior del clasificador)

Tecnológico Antidroga del Departamento de Defensa a través de la Agencia de Investigación Avanzada en Productos de Defensa (DARPA, sigla de su traducción en inglés Defense Advanced Research Products Agency). El objetivo fue desarrollar capacidades de reconocimiento de rostro automático que permitiesen ser utilizadas para ayudar a las fuerzas de seguridad, inteligencia y de leyes, en ejercicio de sus funciones. El corpus de imágenes FERET fue montado para brindar soporte en la prueba y evaluación de algoritmos de reconocimento de rostro mediante pruebas y procedimientos estandarizados. El corpus final consiste de

14051 imágenes de 8 bits en escala de grises de rostros humanos con vistas que varían desde lo frontal hasta perfiles de izquierda y derecha.

Las imágenes faciales fueron recolectadas en 15 sesiones entre Agosto de 1993 y Julio de 1996. Las sesiones de recolección tardaron uno o dos días. Para mantener el grado de consistencia a lo largo de la base de datos, se utilizó la misma configuración física y localización en cada sesión de fotografía. Las imágenes de un individuo se adquirieron en conjuntos de 5 a 11 imágenes. Se tomaron dos tipos de vistas frontales (fa y fb); se requirió una expresión facial diferente para la segunda imagen frontal. Las imágenes restantes fueron obtenidas desde varios ángulos entre los perfiles de derecha a izquierda, tal como se puede ver en la Figura 3.9. Para agregar variaciones a la base de datos, se tomó un segundo conjunto de imágenes, para las cuales se les pidió a los sujetos que se coloquen sus anteojos y/o se acomoden el pelo hacia atrás. El conjunto de imágenes referido como duplicado, indica un segundo conjunto de imágenes de una persona que fueron tomadas en una fecha más tarde, resultando en variaciones de escala, pose, expresión, e iluminación de la cara.

La Figura 3.10 muestra la topología de la red neuronal utilizada en los experimentos, un arreglo de modelos MLP, cada uno asociado a un sujeto a reconocer. En la próxima subsección se mostrará cómo un modelo aMLP que ha aprendido un problema de reconocimiento de rostros con cierta (alta) precisión puede ser comprimido en una representación más compacta utilizando el modelo V-NN propuesto. Nuevamente, esta representación novedosa requiere menos parámetros, manteniendo sin embargo una precisión alta en cuanto al reconocimiento.

3.7.2. Resultados y discusión

Varias topologías MLP han sido evaluadas en este estudio, de las que se extraen los pesos correspondientes de Volterra luego del entrenamiento. Por simplicidad en el análisis de los resultados, en particular con respecto al desempeño de las capacidades de compresión de V-NN en las tablas, se presentan solamente tres clases (c=3) de cada base de datos. Los resultados completos que fueron obtenidos en ambas bases de datos de rostros pueden encontrarse en el Apéndice A.

En los problemas de reconocimiento de rostros, los conjuntos de datos de

entrenamiento y prueba generalmente tienen una alta dimensionalidad debido al tamaño de las fotos. Por lo tanto, como ya se dijo, se necesita un método apropiado de extracción de características. Se puede realizar una representación global a partir de técnicas ampliamente utilizadas tales como las eigenfaces (Turk y Pentland, 1991) mediante la aplicación de PCA para la reducción de la dimensionalidad (Kirby y Sirovich, 1990). A pesar de que PCA puede reducir altamente el espacio de características, la aplicación de una técnica llamada *Scree Test* (Jackson, 1991) reduce aún más la dimensionalidad del espacio de características enfocándose solamente en aquellas eigenfaces más representativas. Scree Test se aplica para determinar el número de componentes principales de los conjuntos de entrenamiento y prueba, de acuerdo a la varianza en los datos. Esta técnica permite obtener las componentes de una manera ordenada de acuerdo a la varianza de los datos con mayor cantidad de información.

Independientemente de los datos que se modelan con PCA, es común utilizar un valor de 85 % del total de la varianza del espacio de características para identificación (Jackson, 1991). Aplicando Scree Test en el conjunto de datos ORL, el 85 % de la varianza del conjunto de entrenamiento se representa mediante 11 eigenfaces, por lo que cada modelo MLP posee 11 neuronas en la capa de entrada. En cuanto al número de neuronas en la capa oculta, se adoptó una simple heurística en la que la cantidad de neuronas en la capa oculta es igual, el doble y el triple de la cantidad de entradas, por lo que N_H es 11, 22 y 33; y en la salida hay una sola neurona que toma el valor de 1 si se reconoce al sujeto. Considerando el caso del conjunto de datos FERET, el 85 % de la varianza se representa mediante 9 eigenfaces. Por lo tanto, cada modelo MLP posee 9 neuronas de entrada, un número de neuronas ocultas que puede ser 9, 18 y 27, y también una sola salida. Para cada señal de entrada que arriba al arreglo, se puede obtener una salida V-NN de un cierto orden. Por ejemplo, para obtener una salida de primer orden, se selecciona el máximo $s^{(1)}$ del arrreglo.

Los parámetros del modelo, pesos y umbrales, se inicializaron con valores aleatorios distribuídos uniformemente entre 0 y 1. Las neuronas en las capas ocultas y de salida poseen funciones de activación sigmoideas. Todos las MLP se entrenaron con el algoritmo Levenberg-Marquardt (Marquardt, 1963) para poder garantizar una convergencia rápida. Se utilizó un procedimiento de validación cruzada con k particiones (Haykin, 2007), con $k = 3$, repitiendo tres veces los

Tabla 3.10: Tasas de reconocimiento (RR_i) para cada clase $i = 1, 2, 3$ para tres clasificadores aV-NN (a=3) y sus correspondientes salidas de primer $s^{(1)}$, segundo $s^{(2)}$ y tercer $s^{(3)}$ orden, para la base de datos de rostros ORL de AT&T Laboratories Cambridge.

	$\mathcal{3}\mathrm{MLP}_{11,11,1}$			$\mathcal{3}\mathrm{MLP}_{11,22,1}$			$\mathcal{3}\mathrm{MLP}_{11,33,1}$		
$\mathrm{RR}_i[\%]$	RR_1	RR_2	RR_3	RR_1	RR_2	RR_3	RR_1	RR_2	RR_3
$a\mathrm{MLP}_{N_I,N_H,N_O}$	100,00	100,00	100,00	100,00	100,00	100,00	100,00	100,00	100,00
$\mathcal{3}\text{V-NN}_{s^{(1)}}$	94,28	93,94	97,47	92,76	92,25	91,92	97,14	91,92	94,28
$\mathcal{3}\text{V-NN}_{s^{(2)}}$	91,92	92,09	91,25	92,76	91,24	94,27	95,29	94,44	90,57
$\mathcal{3}\text{V-NN}_{s^{(3)}}$	91,58	90,23	91,58	90,23	89,06	88,89	86,19	90,74	93,26

Tabla 3.11: Tasas de reconocimiento (RR_i) para cada clase $i = 1, 2, 3$ para tres clasificadores aV-NN (a=3) y sus correspondientes salidas de primer $s^{(1)}$, segundo $s^{(2)}$ y tercer $s^{(3)}$ orden, para la base de datos FERET.

	$\mathcal{3}\mathrm{MLP}_{9,9,1}$			$\mathcal{3}\mathrm{MLP}_{9,18,1}$			$\mathcal{3}\mathrm{MLP}_{9,27,1}$		
$\mathrm{RR}_i[\%]$	RR_1	RR_2	RR_3	RR_1	RR_2	RR_3	RR_1	RR_2	RR_3
$a\mathrm{MLP}_{N_I,N_H,N_O}$	100,00	100,00	100,00	100,00	100,00	100,00	100,00	100,00	100,00
$\mathcal{3}\text{V-NN}_{s^{(1)}}$	94,11	94,27	95,62	86,70	95,62	93,94	88,38	91,41	96, 63
$\mathcal{3}\text{V-NN}_{s^{(2)}}$	91,08	88,38	91,58	88,89	87,88	94,11	92,25	90,07	95, 29
$\mathcal{3}\text{V-NN}_{s^{(3)}}$	90,40	85,35	93, 10	89,56	85,52	85,86	91,92	84,34	85,01

experimentos en cada partición, configurada en forma estándar para que divida las imágenes disponibles de cada persona en un 80 % de los datos de cada clase para entrenamiento y un 20 % para prueba. La estimación de la precisión global de un modelo se calculó promediando las medidas de precisión sobre los conjuntos de datos de prueba, los cuales eran disjuntos. En cada experimento y repetición, no se consideraron los modelos MLP con menos de 100 % de tasa de clasificación para cada clase del conjunto de datos de entrenamiento. Esta restricción sobre el desempeño del clasificador se ha impuesto, como en los experimentos anteriores, con el objeto de poder medir de forma precisa cualquier pérdida de precisión originada por el método de compresión Volterra-NN propuesto.

Para cada modelo $aMLP_{N_I,N_H,1}$ considerado en este estudio, sus correspondientes pesos de Volterra de primer, segundo y tercer orden han sido extraídos de acuerdo al Algoritmo 1 y sus correspondientes salidas $s^{(1)}$, $s^{(2)}$ y $s^{(3)}$ del arreglo de Volterra-NN han sido obtenidas utilizando el Algoritmo 2. La Tabla 3.10 presenta los resultados obtenidos para la tasa de reconocimiento (RR_i) de los modelos para cada clase $i = 1, 2, 3$ sobre la tarea de reconocimiento de rostros, calculada en base al conjunto de prueba de la base de datos ORL; por su parte, la Tabla 3.11

muestra los resultados obtenidos sobre la base de datos FERET.

En la Tabla 3.10 es posible observar que, cuando el clasificador *a*MLP tiene 11 neuronas ocultas ($3\text{MLP}_{11,11,1}$), los valores RR relacionados a la clase 1 (RR_1) están entre 91 % y 94 % para las salidas $3\text{V-NN}_{s^{(1)}}$, $3\text{V-NN}_{s^{(2)}}$ y $3\text{V-NN}_{s^{(3)}}$; para la segunda clase (RR_2), estos valores varían de 90 % a 94 %; y están entre 91 % y aproximadamente 97 % para la tercer clase (RR_3). Con respecto al segundo problema de reconocimiento de rostro (Tabla 3.11) cuando el clasificador *a*MLP tiene 9 neuronas ocultas ($3\text{MLP}_{9,9,1}$), los valores RR son similarmente altos para las clases 1, 2 y 3.

Enfocándose en la Tabla 3.10, es posible observar que se obtiene una tasa RR peor para el problema de clasificación si se utiliza $3\text{V-NN}_{s^{(3)}}$ en lugar del *a*MLP original, particularmente para la clase 2. Este RR bajo es, de todos modos, del 90 %. Por otro lado, la salida $3\text{V-NN}_{s^{(1)}}$ requiere de un número de parámetros significativamente menor que el clasificador original $3\text{MLP}_{11,11,1}$ (una relación de 1:12), logrando además las tasas de reconocimiento más altas. Conclusiones similares se pueden establecer a partir de la Tabla 3.11, en la que se utilizó la base de datos FERET: la salida V-NN más simple es, al mismo tiempo, la mejor.

Pero si se tiene en cuenta que las tres topologías de modelos $3\text{MLP}_{N_I,N_H,1}$, en ambos casos, con $N_I = 11$, $N_H = 11, 22, 33$ para ORL y $N_I = 9$, $N_H = 9, 18, 27$ para FERET, son soluciones para el mismo problema, en realidad no es necesario considerar solamente una topología como la mejor solución. Es decir, hay muchas soluciones posibles al mismo problema, y es posible obtener la mejor de acuerdo a los objetivos que se persiguen. Por ejemplo, la mejor tasa de reconocimiento por clase para la base de datos ORL, se obtiene mediante la salida V-NN de primer orden para $3\text{MLP}_{11,33,1}$, la cual es tan alta como 97,50 %; mientras que la mejor tasa de reconocimiento para el caso de la base de datos FERET se obtiene mediante la salida Volterra-NN de primer orden para $3\text{MLP}_{9,27,1}$, alcanzando una RR de 96,63 % en la clase 3.

Las Tablas 3.12 y 3.13 muestran la tasa de ahorro de espacio para los modelos puestos en discusión en esta parte de resultados, y los valores promedios de la tasa de reconocimiento global para estos modelos. La parte superior de cada tabla muestra el número de parámetros y la medida RR global para las tres topologías *a*MLP. La segunda parte de las tablas muestran parámetros y los valores de la medida de desempeño relacionadas a las salidas del Volterra-NN. En primer lugar,

Tabla 3.12: Comparación de las tasas de reconocimiento global (RR) y de ahorro espacio (SS) para tres clasificadores aV-NN (a=3) y sus correspondientes salidas de primer $s^{(1)}$, segundo $s^{(2)}$ y tercer $s^{(3)}$ orden, para la base de datos de rostros ORL de AT&T Laboratories Cambridge.

		$3MLP_{11,11,1}$		$3MLP_{11,22,1}$		$3MLP_{11,33,1}$	
$P_{aMLP_{N_I,N_H,N_O}} \rightarrow$		432		861		1290	
RR[%] $\rightarrow$		100,00		100,00		100,00	
3V-NN	$P_{as^{(i)}}$	RR[%]	SS[%]	RR[%]	SS[%]	RR[%]	SS[%]
3V-NN$_{s^{(1)}}$	33	95,23	92,36	92,31	96,16	**94,44**	**97,44**
3V-NN$_{s^{(2)}}$	231	91,75	46,53	92,76	73,17	93,43	82,09
3V-NN$_{s^{(3)}}$	594	91,13	−37,50	89,39	31,01	90,07	53,95

se muestra, en la primer fila, el número de parámetros que se necesita para cada arquitectura de clasificador neuronal ($P_{aMLP_{N_I,N_H,N_O}}$) utilizado en este estudio, mientras que, en el primer columna, se muestra el número de pesos de Volterra necesarios para cada modelo Volterra-NN ($P_{as^{(i)}}$). Los valores que completan las columnas RR y SS son las tasas de reconocimiento promedio y las capacidades de ahorro de espacio, respectivamente, para cada combinación entre un clasificador MLP y su correspondiente salida Volterra-NN.

Como se dijo en el marco teórico (ecuaciones (2.33) y (2.34)), el número de parámetros que se necesitan para el clasificador neuronal depende no solamente de N_I sino también del número de neuronas en la capa oculta N_H, como así también del número de elementos en el arreglo. Por lo tanto, considerando la base de datos ORL, el número de parámetros para 3MLP$_{11,11,1}$ es 432, para 3MLP$_{11,22,1}$ es 861, y el modelo 3MLP$_{11,33,1}$ tiene un total de 1290 parámetros (ver parte superior de la Tabla 3.12). En cuanto a la base de datos FERET, el número de parámetros para 3MLP$_{9,9,1}$ es 300, para 3MLP$_{9,18,1}$ es de 597, y el modelo 3MLP$_{9,27,1}$ tiene un total de 894 parámetros (ver Tabla 3.13).

En el caso de las salidas de Volterra-NN de diferentes órdenes, el número de parámetros solamente depende del número de entradas, como se ha mostrado en las ecuaciones (3.12), (3.13) y (3.14). Sin embargo, para un arreglo, este número debe ser multiplicado por el tamaño del arreglo. Por lo tanto el número de parámetros para 3V-NN$_{s^{(1)}}$ es 33, y el número de parámetros necesarios para 3V-NN$_{s^{(2)}}$ y 3V-NN$_{s^{(3)}}$ son 231 y 594, respectivamente. En la base de datos FERET, el número de parámetros para 3V-NN$_{s^{(1)}}$ es 27, y el número de parámetros necesarios para 3V-NN$_{s^{(2)}}$ y 3V-NN$_{s^{(3)}}$ son 162 y 405, respectivamente.

Tabla 3.13: Comparación de las tasas de reconocimiento global (RR) y de ahorro espacio (SS) para tres clasificadores aV-NN (a=3) y sus correspondientes salidas de primer $s^{(1)}$, segundo $s^{(2)}$ y tercer $s^{(3)}$ orden, para la base de datos de rostros FERET.

		$3MLP_{9,9,1}$		$3MLP_{9,18,1}$		$3MLP_{9,27,1}$	
$P_{aMLP_{N_I,N_H,N_O}} \rightarrow$		300		597		894	
RR[%] $\rightarrow$		100,00		100,00		100,00	
3V-NN	$P_{as^{(i)}}$	RR[%]	SS[%]	RR[%]	SS[%]	RR[%]	SS[%]
3V-NN$_{s^{(1)}}$	27	94,67	91,00	92,09	95,48	**92,14**	**96,98**
3V-NN$_{s^{(2)}}$	162	90,35	46,00	90,29	72,86	92,54	81,88
3V-NN$_{s^{(3)}}$	405	89,62	−35,00	86,98	32,16	87,09	54,70

Enfocándose en la Tabla 3.12, es posible ver que cuando se utiliza la salida 3V-NN$_{s^{(1)}}$ en lugar del clasificador 3MLP$_{11,11,1}$, es posible obtener una tasa de reconocimiento de 95,23 % para el problema de reconocimiento de rostro, y al mismo tiempo se puede alcanzar una tasa de compresión o ahorro de espacio del 92,36 %. Aproximadamente la mitad de esta tasa de ahorro de espacio es obtenida si se utiliza la salida 3V-NN$_{s^{(2)}}$, y no hay compresión cuando se utiliza 3V-NN$_{s^{(3)}}$. Para el modelo 3MLP$_{11,22,1}$, la compresión que se alcanza por los modelos Volterra-NN es más alta debido a la cantidad de parámetros asociados con el modelo; en este caso, los valores SS alcanzan un máximo de 96,16 % cuando se considera la menor cantidad posible de parámetros (una salida Volterra-NN de primer orden). Un caso similar es el relacionado al modelo 3MLP$_{11,33,1}$, en el que esta tendencia también se verifica. Los resultados obtenidos con el otro conjunto de datos son bastante similares (Tabla 3.13). Por ejemplo, se puede obtener una tasa de reconocimiento de 94,67 % y se puede alcanzar una tasa de compresión o ahorro de espacio de 91 % cuando se utiliza la salida 3V-NN$_{s^{(1)}}$ en lugar del clasificador 3MLP$_{9,9,1}$.

Para ambas Tablas 3.12 y 3.13 es posible concluir que un arreglo de MLPs para tareas de clasificación complejas puede ser correctamente comprimida utilizando el modelo Volterra-NN propuesto en esta tesis; y esta tasa de compresión puede ser, en algunos casos, mayor al 90 %. Más aún, en general, se alcanzan tasas de reconocimiento muy altas. De hecho, la arquitectura del modelo aMLP puede ser más y más compleja, puede tener muchas más unidades ocultas, pero el número de pesos necesarios para construir cada salida de Volterra-NN seguirá siendo la misma mientras el número de variables de entrada de los problemas sean las

mismas, y esto será reflejado ciertamente en tasas de ahorro de espacio aún más altas. Inclusive, en algunos casos, estas altas tasas de compresión no deben ser pagadas con bajas tasas de clasificación. Por ejemplo, para el modelo $3\text{MLP}_{11,11,1}$ en la base de datos ORL, el mejor valor RR está asociado a la salida $3\text{V-NN}_{s^{(1)}}$, alcanzando un 95,23 %, y obteniendo una tasa de ahorro de espacio de más del 92 %. Exactamente el mismo caso puede observarse en la base de datos FERET, en el que el modelo $3\text{MLP}_{9,9,1}$ alcanza un RR de 94,67 %, obteniendo una tasa de ahorro de espacio del 91 %.

Considerando el caso $3\text{MLP}_{11,22,1}$, en la Tabla 3.12, el mejor valor RR está asociado con la salida $3\text{V-NN}_{s^{(2)}}$, la que tiene solamente una SS de 73,17 %. Pero si se necesita una mejor compresión para esta topología, es posible elegir la $3\text{V-NN}_{s^{(1)}}$, que tiene una SS de 96,16 % y preserva un valor RR similar de 92,31 %. A pesar de que las diferencias entre las salidas $3\text{V-NN}_{s^{(1)}}$ y $3\text{V-NN}-_{s^{(2)}}$ son mínimas teniendo en cuenta el segundo modelo en ambas tablas, en la Tabla 3.13 los resultados son ligeramente diferentes si se considera el modelo $3\text{MLP}_{9,18,1}$. Aquí, el mejor RR y SS se obtiene en el mismo caso, alcanzando un ahorro de espacio de 95,48 % preservando el 92,09 % de habilidad de reconocimiento si se escoge la salida $3\text{V-NN}_{s^{(1)}}$.

Para el modelo $3\text{MLP}_{11,33,1}$, en la Tabla 3.12, el mejor valor RR es asociado con el modelo Volterra-NN $3\text{V-NN}_{s^{(1)}}$ y es posible alcanzar un SS de 97,44 %. Este es un resultado muy interesante, dado que con aproximadamente menos del 3 % de los parámetros del clasificador aMLP original, la capacidad de clasificación de $3\text{V-NN}_{s^{(1)}}$ es muy alta (94,44 %) y cercana al modelo aMLP original. Este caso particular puede considerarse como el mejor resultado global obtenido en este caso de estudio, en el que el clasificador $3\text{MLP}_{11,33,1}$ puede comprimirse en aproximadamente un 98 % utilizando la salida Volterra-NN $3\text{V-NN}_{s^{(1)}}$ correspondiente, sin pérdida significativa de capacidad de reconocimiento. Es posible obtener conclusiones similares en la Tabla 3.13. Una vez más, una pequeña diferencia de 0.4 % puede observarse asociada a la salida ($3\text{V-NN}_{s^{(1)}}$ o $3\text{V-NN}_{s^{(2)}}$) que está relacionada con el mejor RR obtenido en el clasificador $3\text{MLP}_{9,27,1}$. A pesar de esto, es posible afirmar que el mejor resultado global obtenido en este caso está asociado con la salida Volterra-NN $3\text{V-NN}_{s^{(1)}}$, en cual el clasificador puede comprimirse en casi un 97 %, manteniendo un 92 % de tasa de reconocimiento.

En este caso de estudio más complejo, el modelo V-NN propuesto ha sido

Tabla 3.14: RR con la misma SS en aV-NN, OBD, u OBS.
Mejor valor en negrita.
(Base de datos ORL).

		$\mathcal{3}MLP_{11,11,1}$	
$P_{aMLP_{I,H,O}} \rightarrow$		432	
RR[%] $\rightarrow$		100,00	
	$P_{as^{(i)}}$	RR[%]	SS[%]
$\mathcal{3}$V-NN$^{(1)}$		**95,23**	
OBD	33	40,57	92,36
OBS		61,11	

Tabla 3.15: SS en aV-NN, OBD, y OBS, con la misma RR.
Mejor valor en negrita.
(Base de datos ORL).

		$\mathcal{3}MLP_{11,11,1}$	
$P_{aMLP_{I,H,O}} \rightarrow$		432	
RR[%] $\rightarrow$		100,00	
	$P_{as^{(i)}}$	RR[%]	SS[%]
$\mathcal{3}$V-NN$^{(1)}$	33		**92,36**
OBD	192	≈ 95	58,44
OBS	285		34,03

Tabla 3.16: RR con la misma SS en aV-NN, OBD, y OBS.
Mejor valor en negrita.
(Base de datos FERET).

		$\mathcal{3}MLP_{9,9,1}$	
$P_{aMLP_{I,H,O}} \rightarrow$		300	
RR[%] $\rightarrow$		100,00	
	$P_{as^{(i)}}$	RR[%]	SS[%]
$\mathcal{3}$V-NN$^{(1)}$		**94,67**	
OBD	27	38,38	91,00
OBS		40,24	

Tabla 3.17: SS en aV-NN, OBD, y OBS, con la misma RR.
Mejor valor en negrita.
(Base de datos FERET).

		$\mathcal{3}MLP_{11,11,1}$	
$P_{aMLP_{I,H,O}} \rightarrow$		300	
RR[%] $\rightarrow$		100,00	
	$P_{as^{(i)}}$	RR[%]	SS[%]
$\mathcal{3}$V-NN$^{(1)}$	27		**91,00**
OBD	204	≈ 95	32,00
OBS	198		34,00

comparado contra dos algoritmos clásicos de poda de pesos, considerados como una forma simple de compresión para modelos MLP, tales como OBD (Cun y otros, 1990) y OBS (Hassibi y otros, 1993). En esta comparación, en las Tablas 3.14 a 3.17 se han evaluado dos aspectos: i) la RR de cada método mientras se mantiene el mismo número de parámetros (Tablas 3.14 y 3.16, mejor valor en negrita); ii) la SS considerando aproximadamente el mismo valor RR para los tres métodos (Tablas 3.15 y 3.17, mejor valor en negrita). En el caso de la base de datos ORL, para los 33 parámetros utilizados para $\mathcal{3}$V-NN$_{s^{(1)}}$ con el objeto de alcanzar una RR de 95,23 %, OBS y OBD obtuvieron valores RR de 40,57 % y 61,11 %, respectivamente (ver Tabla 3.14). Considerando la base de datos FERET, dado que $\mathcal{3}$V-NN$_{s^{(1)}}$ necesita 27 parámetros para un valor RR de 94,67 %, OBD obtuvo un valor RR de 38,38 %, y OBS alcanzó un RR de 40,24 % para el mismo número de parámetros (ver Tabla 3.16). Con respecto a la compresión, como es posible ver en las Tablas 3.15 y 3.17, para mantener una RR de 95 % en ambas bases de datos, mientras V-NN necesita aproximadamente 30 parámetros, OBS y OBD

requieren aproximadamente 200 parámetros o más.

De los análisis anteriores puede verse que es necesario observar ambas tablas de RR y SS, para cada conjunto de datos, para poder seleccionar el modelo mejor comprimido, lo que puede ser confuso también. Es por esto que se propone a continuación este análisis de una forma más simple y directa, a través del uso de la medida ∂ que ha sido introducida en la Sección 3.4, con un estudio más detallado del pesado de las diferentes soluciones mediante diferentes valores ∂.

Las Tablas 3.18 y 3.19 presentan los valores ∂ para cada *3*V-NN, considerando tres valores posibles de γ. Las filas agrupan las tres posibles salidas *3*V-NN ($s^{(1)}$, $s^{(2)}$ y $s^{(3)}$) para cada topología *3*MLP que se analiza. Las últimas tres columnas principales representan la aplicación de tres valores diferentes de γ, cada uno para enfatizar la prioridad de SS por sobre RR ($\gamma = 0{,}25$), RR sobre SS ($\gamma = 0{,}75$), o ambos equitativamente pesados ($\gamma = 0{,}5$).

Teniendo en cuenta los resultados en la columna $\gamma = 0{,}25$ de la Tabla 3.18 por un lado, es posible concluir que la solución de mejor compromiso es alcanzada por *3*V-NN$_{s^{(1)}}$ cuando la topología del modelo MLP es *3*MLP$_{11,33,1}$, alcanzando el mínimo valor ∂ de 0,024 (esto corresponde al mejor SS de 97,44 %, y su RR llega a 94,44 %). Por el otro lado, enfocándose en la misma columna pero de la Tabla 3.19, se pueden establecer conclusiones similares debido a que la mejor solución es alcanzada también por *3*V-NN$_{s^{(1)}}$, cuando el modelo MLP es *3*MLP$_{9,27,1}$, con el mínimo valor ∂ de 0,030 (SS es 96,98 % mientras que su RR es 92,14 %).

Para la columna $\gamma = 0{,}5$, la mejor solución de compromiso está asociada con la misma solución que en el caso previo: *3*V-NN$_{s^{(1)}}$ para el modelo MLP *3*MLP$_{11,33,1}$ en la Tabla 3.18 y para el modelo MLP *3*MLP$_{9,27,1}$ en la Tabla 3.19. Para este caso, ambas medidas de desempeño RR y SS en ambas experiencias tienen valores altos y son la solución de compromiso cuando no hay prioridad de criterios entre las tasas de RR y SS.

En la columna en la que $\gamma = 0{,}75$, el modelo *3*V-NN$_{s^{(1)}}$ para el clasificador *3*MLP$_{11,11,1}$ es la mejor solución en la Tabla 3.18, alcanzando un mínimo valor ∂ de 0,041. Su valor RR es el mejor (95,23 %) y su SS llega a 92,36 %. En la Tabla 3.19, el modelo *3*V-NN$_{s^{(1)}}$ para el clasificador *3*MLP$_{9,9,1}$ es también la mejor solución para la tercer columna, con un valor RR de 94,67 % y SS de 91 %.

Es necesario notar que en la Tabla 3.18 el mejor valor es muy cercano al valor $\partial = 0{,}042$ para *3*V-NN$_{s^{(1)}}$ cuando la topología del modelo MLP es *3*MLP$_{11,33,1}$.

Tabla 3.18: Comparación de diferentes soluciones de compromiso ∂ para las salidas del modelo *3*Volterra-NN correspondiente a un clasificador *3*MLP, para la base de datos de rostros ORL de AT&T Laboratories Cambridge. El mejor valor para cada modelo es resaltado en negrita.

∂		$\gamma_1 = 0{,}25$	$\gamma_2 = 0{,}5$	$\gamma_3 = 0{,}75$
$3\text{MLP}_{11,11,1}$	$3\text{V-NN}_{s(1)}$	0,059	0,045	**0,041**
	$3\text{V-NN}_{s(2)}$	0,402	0,271	0,147
	$3\text{V-NN}_{s(3)}$	1,031	0,689	0,350
$3\text{MLP}_{11,22,1}$	$3\text{V-NN}_{s(1)}$	0,035	0,043	0,058
	$3\text{V-NN}_{s(2)}$	0,202	0,139	0,086
	$3\text{V-NN}_{s(3)}$	0,518	0,349	0,190
$3\text{MLP}_{11,33,1}$	$3\text{V-NN}_{s(1)}$	**0,024**	**0,031**	0,042
	$3\text{V-NN}_{s(2)}$	0,135	0,095	0,067
	$3\text{V-NN}_{s(3)}$	0,346	0,236	0,137

Tabla 3.19: Comparación de diferentes soluciones de compromiso ∂ para las salidas del modelo *3*Volterra-NN correspondiente a un clasificador *3*MLP, para la base de datos de rostros Facial Recognition Technology (FERET). El mejor valor para cada modelo es resaltado en negrita.

∂		$\gamma_1 = 0{,}25$	$\gamma_2 = 0{,}5$	$\gamma_3 = 0{,}75$
$3\text{MLP}_{9,9,1}$	$3\text{V-NN}_{s(1)}$	0,069	0,052	**0,046**
	$3\text{V-NN}_{s(2)}$	0,406	0,274	0,153
	$3\text{V-NN}_{s(3)}$	1,013	0,677	0,346
$3\text{MLP}_{9,18,1}$	$3\text{V-NN}_{s(1)}$	0,039	0,046	0,060
	$3\text{V-NN}_{s(2)}$	0,205	0,144	0,100
	$3\text{V-NN}_{s(3)}$	0,510	0,345	0,196
$3\text{MLP}_{9,27,1}$	$3\text{V-NN}_{s(1)}$	**0,030**	**0,042**	0,059
	$3\text{V-NN}_{s(2)}$	0,137	0,098	0,072
	$3\text{V-NN}_{s(3)}$	0,341	0,236	0,149

Pero, en este último caso, SS es más alta que RR y esta situación es penalizada mediante el parámetro γ, debido a las prioridades en la selección del modelo. Es importante notar que para todas las opciones previamente analizadas, $3\text{V-NN}_{s(1)}$ es el modelo de mejor compromiso, en general, el que requiere pocos parámetros para ser representado y, por lo tanto, es el más pequeño.

Otro punto importante a resaltar es que el paso de selección de modelo es una tarea necesaria que debe realizarse con el objeto de determinar con certeza cuál modelo y qué parámetros son los más adecuados para el conjunto de datos bajo estudio. En este sentido es que la medida ∂ puede realmente ayudar en la comparación entre modelos. Más aún, esta medida puede encontrar el mejor clasificador posible para cada clase, combinando salidas Volterra-NN de diferentes órdenes, obtenidos de diferentes topologías y configuraciones neuronales.

Finalmente, a partir del análisis de las Tablas 3.18 y 3.19, es posible observar que se obtienen resultados consistentes con respecto a los análisis detallados desarrollados en Tabla 3.12 y 3.13, alcanzados sin embargo en una manera más compacta y simple, gracias a la nueva medida para selección de modelo propuesta.

3.8. Resumen

En este capítulo se ha presentado un nuevo enfoque para la compresión de una red neuronal enfocado sobre un problema de clasificación. Se ha mostrado un método para obtener una representación compacta de un modelo RNA utilizando el modelo Volterra-NN y sus correspondientes salidas de diferente orden. Ha sido presentado también un método para obtener una representación compacta de un arreglo de MLPs utilizando salidas de diferente orden del modelo aV-NN.

Han sido explicados, también, dos algoritmos que implementan el enfoque antes mencionado. El Algoritmo 1 permitió extraer los pesos de Volterra a partir de los parámetros del modelo MLP, luego de haber sido entrenado. El Algoritmo 2 permitió obtener las salidas de Volterra-NN utilizando los pesos de Volterra cuando se recibe un nuevo punto a ser clasificado.

Además, se ha propuesto una medida para la selección de modelos, que permite considerar la calidad de un modelo por sobre otro de una forma más simple. Esta nueva medida ∂ ha permitido evaluar las diferentes soluciones de manera compacta considerando solo un valor, que ha surgido de la relación pesada entre la tasa de reconocimiento y el ahorro de espacio.

Todos los resultados experimentales, obtenidos en primer lugar sobre el problema de clasificación Iris, han resaltado las capacidades de compresión del modelo Volterra-NN simple, permitiendo mantener en algunos casos la misma tasa de reconocimiento que el modelo MLP original, y alcanzando al mismo tiempo tasas de compresión muy altas.

Similarmente, se ha demostrado que esta capacidad de ahorro de espacio no se pierde cuando se busca comprimir un modelo arreglo de MLP a través del modelo aV-NN. Tanto para el caso de estudio de los problemas de clasificación más complejos, basados en conjunto de datos conocidos en la literatura, como el del problema biométrico real de reconocimiento de rostros, se ha probado que el modelo propuesto es capaz de resolver estos problemas efectiva y eficientemente,

obteniendo casi la misma precisión que las configuraciones de los arreglos de clasificadores MLP. Más aún, estos arreglos de clasificadores MLP han sido objeto de una significativa compresión a modelos más simples, permitiendo utilizar menor cantidad de parámetros para obtener, prácticamente, los mismos resultados de desempeño.

Los resultados presentados en este capítulo han sido publicados en un congreso internacional con referato (Rubiolo y otros, 2010), en un congreso nacional con referato (Rubiolo, 2012), y una publicación en revista indexada con Factor de Impacto 2012 de 1.168 (Rubiolo y otros, 2013b).

CAPÍTULO 4

Clasificador neuronal para correspondencia entre ontologías

4.1. Introducción

Este capítulo presenta un clasificador basado en un modelo Red Neuronal Artificial (RNA), utilizado para definir la operación de correspondencia entre ontologías (OM, sigla de su traducción en inglés Ontology Matching). En primer lugar, se explica el problema de la correspondencia ontológica (Sección 4.2). Luego se explica el método propuesto que tiene en cuenta tanto la información a nivel de esquema como a nivel de instancia, a partir de las ontologías y las anotaciones semánticas (Sección 4.3). El mismo será probado en un caso de estudio real, comparando los resultados obtenidos con los de un algoritmo de OM reconocido en la literatura (Sección 4.4). Éstos han sido comunicados en un congreso internacional con referato (Rubiolo y otros, 2009) y publicados en una revista indexada con Factor de Impacto 2012 de 3.643 (Rubiolo y otros, 2012).

4.2. El problema de la correspondencia entre ontologías

Las Ontologías se han convertido en un estándar de facto para la descripción semántica de los datos, recursos y servicios, en grandes sistemas distribuidos tales como la Web (Badica y otros, 2010). Es decir, proveen características útiles para los sistemas inteligentes, para la representación del conocimiento en general, y para los proceso de ingeniería de conocimiento (Gomez-Perez y otros, 2005). En la literatura hay diferentes definiciones sobre lo que es una ontología y sus

diferentes clasificaciones (Staab y Studer, 2009). En este trabajo, una *ontología de dominio* es considerada como un artefacto representacional de la semántica de un dominio dado. Un *dominio* es una porción de la realidad que conforma el objeto de una ciencia o tecnología (Smith y otros, 2006). Las unidades representacionales de una ontología son las siguientes: *términos*, *relaciones*, *propiedades*, *axiomas* e *instancias*. Ya que las ontologías de dominio proveen una conceptualización compartida de cierto dominio, ellas pueden ser utilizadas para describir la semántica de los recursos mediante el agregado de metadatos.

Con el objeto de definir la semántica de un contenido digital, es necesario formalizar las ontologías utilizando lenguajes específicos tales como RDF (sigla de su nombre en inglés Resource Description Framework) y OWL (sigla de su nombre en inglés Web Ontology Language) (Smith y otros, 2004). Mientras que RDF es un lenguaje de propósito general para representar información sobre los recursos en la Web, OWL es un lenguaje de marcado (o etiquetado) semántico para publicar y compartir ontologías. Aunque RDF fue originalmente pensado para representar el metadato de los recursos de la web, puede también utilizarse para representar información sobre objetos que pueden ser identificados en la Web. La construcción básica en RDF es una tripleta *(Sujeto,Propiedad,Valor)*: un sujeto S tiene una propiedad P con un valor V. Una tripleta RDF corresponde a la relación que puede ser escrita como *(S,P,V)*, por ejemplo *(http://www.books.org/ISBN0012515866,hasPrice,62)*; y *(Professor,teachesSubject,Artificial Intelligence)* en el caso de un sitio web de una Universidad que está anotado en una ontología.

La existencia de múltiples y heterogéneas ontologías de dominio específico, y su desarrollo distribuído, introduce un problema: la coexistencia de muchas ontologías desarrolladas independientemente, describiendo el mismo campo de conocimiento o uno muy similar. Estas ontologías, o son idénticas, o presentan diferencias menores, tales como convenciones de diferente nombre o estructuras diferentes en la forma de representar el conocimiento. Cualquier aplicación que involucra múltiples ontologías debe establecer una correspondencia semántica entre ellas para asegurar la interoperabilidad. Ejemplos de tales aplicaciones surgen en muchos dominios, incluídos el e-commerce, e-learning, la extracción de información, la bioinformática, los servicios web, el turismo, entre otros (Staab y Studer, 2009). Por esta razón, es necesaria la OM para resolver el problema.

La OM tiene por objetivo encontrar las correspondencias existentes entre elementos semánticamente relacionados de diferentes ontologías. Las correspondencias pueden presentarse para las equivalencias, como así también para otras relaciones, tales como una consecuencia, subsunción, o disjunción, entre los elementos. Por esta razón, el problema de cómo integrar ontologías de dominio heterogéneas en la web puede ser finalmente resuelto utilizando OM (Davies y otros, 2007).

Un proceso de OM puede ser formalmente definido como:

$$(S, P, V) \rightarrow ((O_1, c_1), \cdots, (O_i, c_i)) \tag{4.1}$$

donde (S,P,V) representa la tripleta (sujeto, propiedad, valor), i es el número de las ontologías consideradas que pertenecen al mismo campo de conocimiento, y c_i representa el resultado del proceso de asociación, el cual es un valor de asociación númerico entre 0 y 1 para cada ontología O_i considerada.

Existen diferentes algoritmos para implementar el proceso de correspondencia, el que puede generalmente ser clasificado a lo largo de dos dimensiones (Davies y otros, 2007). Por un lado, existe una distinción entre una correspondencia basada en esquemas y la basada en instancias. Un modelo de correspondencia basado en esquemas usa diferentes aspectos de los conceptos y relaciones en las ontologías, y aplica una medida de similaridad para determinar la correspondencia. Un modelo de correspondencia basado en instancias toma las instancias de concepto y las compara para descubrir similaridad. Por el otro lado, hay una distinción entre una correspondencia a nivel elemento y una a nivel de estructura. Mientras que un modelo de correspondencia a nivel de elemento compara propiedades de un concepto o la relación específica para encontrar similaridades, un asociador a nivel estructura compara la estructura de las ontologías para encontrar similaridades. Estos asociadores pueden también combinarse entre sí (Euzenat y Shvaiko, 2007).

Las técnicas de OM, esencialmente, identifican la afinidad semántica entre conceptos pertenecientes a diferentes ontologías. Entre las propuestas recientes, los métodos de aprendizaje maquinal pueden procesar el problema de correspondencia a través de la presentación de muchos ejemplos correctos (positivos) e incorrectos (negativos). Tales algoritmos requieren datos de ejemplo a través de los cuales aprender. Utilizando aprendizaje maquinal, los modelos de correspondencia operan usualmente en dos fases: (i) la fase de aprendizaje o entrenamiento,

y (ii) la fase de clasificación o de correspondencia. Durante la primera fase, se crean los datos de entrenamiento para el proceso de aprendizaje, por ejemplo a través de la asociación manual de dos ontologías. Durante la segunda fase, el modelo de correspondencia entrenado es utilizado para hacer corresponder nuevas ontologías. El aprendizaje puede procesarse en línea (on-line), es decir, el sistema puede aprender contínuamente, o fuera de línea (off-line), por lo que su velocidad no es tan relevante como su precisión. Usualmente, este proceso es llevado a cabo dividiendo el conjunto de datos disponible. Por ejemplo, considerando un conjunto de ejemplos de alineaciones positivas y negativas, los ejemplos serían divididos en un conjunto de entrenamiento y un conjunto de prueba, los cuales podrían ser utilizados para la evaluación del desempeño del algoritmo de aprendizaje (Euzenat y Shvaiko, 2007).

En respuesta al desafío de la OM en diversos contextos de aplicación, han aparecido varias propuestas, las cuales aplican técnicas de aprendizaje maquinal para crear asociaciones semánticas. Por ejemplo, herramientas basadas en RNA, propuestas en la literatura, utilizan información relacionada a esquemas e instancias de las ontologías a fin de producir reglas para la integración ontológica en bases de datos heterogéneas (Euzenat y Shvaiko, 2007). Las RNAs han sido utilizadas no solamente para descubrir las correspondencias entre atributos mediante la categorización y clasificación (Li y Clifton, 1995) o aprendiendo parámetro de asociación, tales como los pesos de asociación, sino que también para poner a punto los sistemas de asociación con respecto a una tarea de asociación particular (Ehrig, 2007).

Dada la información a nivel esquema y a nivel instancia, a veces es útil agrupar estos datos en categorías para disminuir la complejidad computacional de las manipulaciones adicionales de datos. Las redes neuronales de tipo mapas auto-organizativos pueden utilizarse para este propósito, ya que las neuronas en la red se organizan a sí mismas de acuerdo con las características de los patrones de entrada. Este tipo de modelos neuronales son utilizados en el asociador ontológico de (Curino y otros, 2007), el que utiliza un modelo de red neuronal para resolver inconsistencias semánticas.

Durante el proceso de correspondencia, diferentes aspectos semánticos, tales como los nombres, propiedades y relaciones de concepto, contribuyen en diferente grado al resultado de la asociación como un vector de pesos. Cómo aprenderlos no

es una tarea trivial, y el trabajo de investigación actual depende en gran medida de heurísticas humanas. En (Huang y otros, 2008) un modelo RNA aprende y ajusta los pesos para evitar algunas de las desventajas del enfoque de OM basado en reglas como así también del basado en el aprendizaje, los cuales ignoran la información que los datos de instancia pueden proveer. Similarmente, el trabajo de (Chortaras y otros, 2005) usa instancias para aprender las similaridades entre conceptos ontológicos, y así crear luego una nueva ontología combinada. En (Mao y otros, 2010) se propone un enfoque adaptativo para encontrar las correspondencias semánticas entre elementos similares de ontologías diferentes. Este enfoque mide primero las similaridades linguísticas y estructurales de las ontologías en un modelo de espacio vectorial, y luego utiliza un estimador neuronal para aquellas similaridades.

En (Jung, 2010) se presenta una estrategia de OM basada en instancia para dar soporte a la interoperabilidad automática entre sistemas de información basados en ontologías en ambientes distribuídos. Hay algunos métodos para inducir estructuralmente las relaciones de clases a partir de las individuales. Éstas están basadas generalmente en técnicas internas y basadas en cadenas, sin considerar la estructura de la ontología (Euzenat y Shvaiko, 2007). La mayoría de estos métodos fueron creados para dos propósitos principales: (i) integrar diferentes bases de datos, tales como Automatch (Berlin y Motro, 2002), iMAP (Doan y Halevy, 2005) y Dumas (Bilke y Naumann, 2005); y (ii) para fusionar dos ontologías que comparten el mismo conjunto de instancias, tales como FCA-Merge (Stumme y Maedche, 2001). Sin embargo, la mayoría de las herramientas de OM confían en las heurísticas que detectan alguna clase de similaridad en la descripción de los conceptos, y la estructura del grafo ontológico. Ya que estas herramientas trabajan generalmente a nivel terminológico de las ontologías sin tener en cuenta sus instancias, no son apropiadas para el escenario estudiado en este capítulo.

Los algoritmos H−Match (Castano y otros, 2006) y GLUE (Doan y otros, 2003) (Doan y otros, 2004) están cercanamente relacionados al modelo propuesto en este capítulo. Sin embargo, los sistemas GLUE no trabajan muy bien si las instancias difieren sintácticamente. Más aún, no pueden determinar una asociación si existe una falta de instancias. Por esta razón, se ha elegido solamente el algoritmo H-Match para comparar esta propuesta, y los resultados obtenidos se mostrarán en la Sección 4.4.2

El modelo propuesto en este capítulo está basado en redes neuronales, teniendo en cuenta la información a nivel de esquema y de instancia. El modelo se centra en la hipótesis de que las etiquetas son identificadores humanos (nombres) para las instancias, las que son normalmente compartidas por una comunidad con un lenguaje en común dentro de un campo específico de conocimiento. Por lo tanto, se puede afirmar que, si las etiquetas son las mismas, es probable que las instancias asociadas a ellas sean las mismas, o al menos estén semánticamente relacionadas (Ehrig, 2007). Un término en una ontología, sin embargo, puede definirse también como representativo de un grupo de instancias. Es posible, por lo tanto, inferir que los términos que poseen las mismas instancias son los mismos, como así también que las instancias que tienen el mismo término madre (etiqueta) son similares. Se propone codificar el conocimiento ontológico como entradas a un modelo neuronal, independientemente de su idioma, combinado con un modelo basado en RNA entrenado como un clasificador para la asociación de un término y una ontología.

4.3. Modelo de correspondencia ontológica basado en RNA

El objetivo de esta sección es describir cómo el modelo de redes neuronales propuesto es desarrollado, entrenado y finalmente utilizado para realizar OM.

4.3.1. Escenario motivador

En (Stegmayer y otros, 2007), se propuso una arquitectura para el descubrimiento de fuentes de conocimiento en la web, que está compuesta de agentes móbiles, el agente de descubrimiento de fuente de conocimiento KSD (sigla de su traducción al inglés Knowledge Source Discovery) y nodos web. Los agentes móbiles reciben una solicitud de un usuario final y, utilizando una lista generada por el KSD, visitan los nodos buscando una respuesta. El agente KSD es responsable de conocer qué nodos pueden proveer el conocimiento dentro de un área específica, y de indicar una ruta a los agentes móviles que acarrean la solicitud del usuario. El agente KSD conoce la ubicación (URL) de los nodos que pueden brindarle el conocimiento, pero en realidad no puede proveer los archivos,

imágenes, documentos, etc., que son almacenados en los nodos.

Los otros componentes de la arquitectura son nodos. Cada nodo tiene su propia ontología utilizada para marcar semánticamente la información publicada en sus sitios web. Supongamos que hay cuatro nodos (A, B, C y D) que pertenecen al campo de conocimiento de Investigación y Desarrollo (I+D), tal como se muestra en la Figura 4.1. Los nodos A y C utilizan la ontología KA[1] , mientras que el nodo B utiliza la ontología SWRC[2] (sigla de su nombre en ingles Semantic Web for Research Communities), y finalmente, el nodo D utiliza la ontología OLID[3] . Todas estas ontologías fueron implementadas en el lenguaje OWL. Como puede observarse, cada nodo puede utilizar una ontología diferente para anotar semánticamente la información que provee, aún si éstas pertenecen al mismo campo de conocimiento. Más aún, pueden variar no solamente en la forma en que representan un concepto, sino que también en su idioma, por ejemplo, Inglés, Español o ambos.

RDF es utilizado para definir una marcación semántica basada en una ontología para cada nodo. Cada tripleta RDF asigna entidades y relaciones al texto, las que están enlazadas a sus descripciones semánticas en una ontología. Por ejemplo, en el Nodo A, las siguientes tripletas: <*O.C., interest, Semantic Grid*>,<*O.C., interest, Semantic Web*> y <*O.C.,interest,Web Services*> representan los intereses de investigación de O.C. descriptos en el texto, tal como puede observarse a la izquierda de la Figura 4.1.

El agente KSD debe ser capaz de identificar dinámicamente qué nodo puede satisfacer una solicitud (tripleta de consulta) que trae hacia sí un agente móvil. Esto requiere modelos y técnicas que permitan identificar los conceptos ontológicos que poseen afinidad semántica entre ellos, aún cuando sean sintácticamente diferentes. Para resolver este problema de OM, se propone el uso de un modelo RNA con aprendizaje supervisado almacenado en la base de conocimiento del agente KSD.

[1] http://protege.cim3.net/file/pub/ontologies/ka/ka.owl
[2] http://ontoware.org/projects/swrc/
[3] http://libertad.univalle.edu/jsequeda/ontology/olid.owl

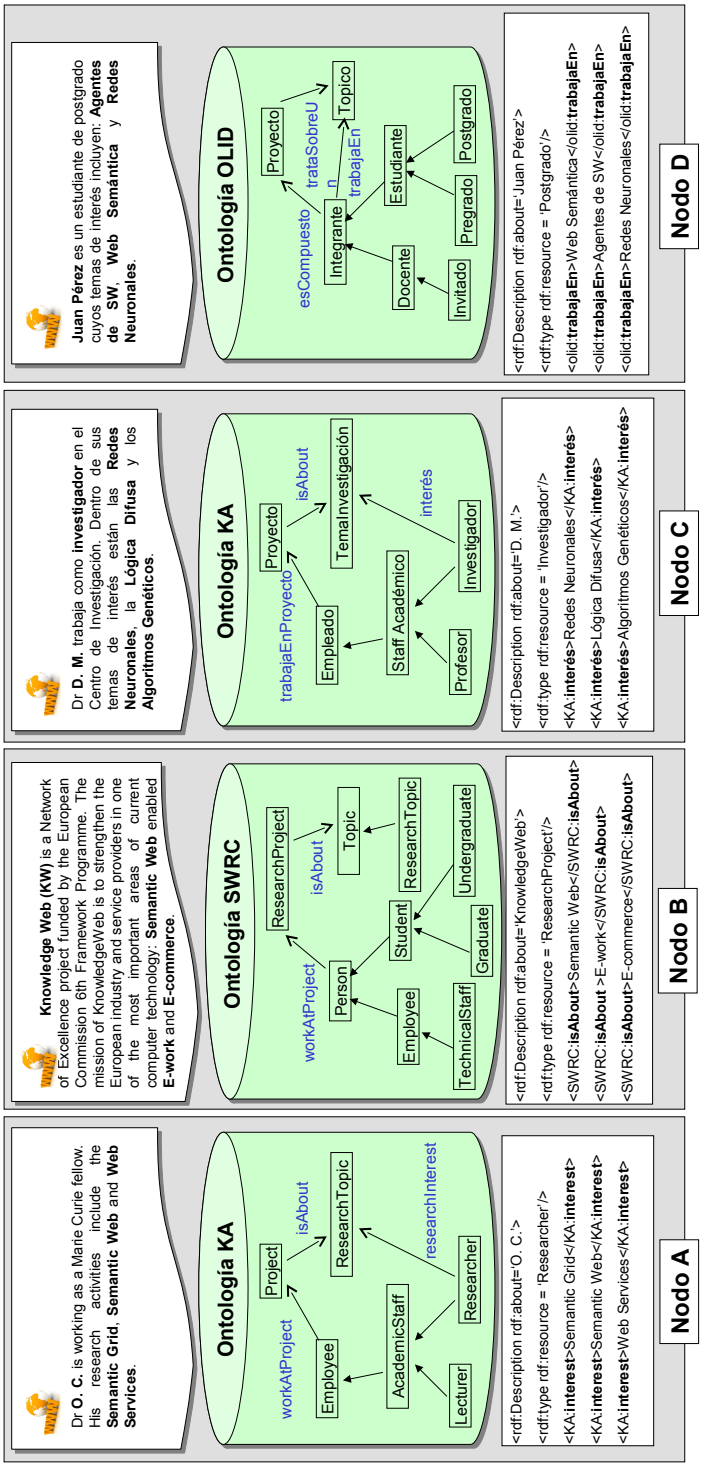

Figura 4.1: Dominios pertenecientes al campo I+D y sus anotaciones semánticas. A partir de una página web (parte superior de la figura) y una ontología de dominio (parte medio) se obtienen las anotaciones semánticas en tripletas RDF (parte inferior)

4.3.2. Definición del modelo.

De acuerdo a la clasificación de los métodos de asociación presentados en la sección anterior, el asociador basado en RNA propuesto aquí utiliza información a nivel esquema, la cual se extrae de las ontologías de dominio, como así también de información de nivel instancia, la que es extraída de las anotaciones RDF basadas en estas ontologías de dominio.

Para las redes neuronales, un problema de asociación puede ser visto como un problema de clasificación. En primer lugar, un asociador basado en RNA utiliza la información a nivel esquema e instancia, asociada con la ontología X para aprender un clasificador para el nodo X y sus correspondientes tripletas o anotaciones, y luego utiliza la información de los niveles esquema e instancia asociada con la ontología Y para aprender un clasificador para el nodo Y. Luego, ambos clasificadores son consultados con tripletas de consulta. La misma idea se aplica a más de dos nodos. Con el objeto de construir un clasificador para el nodo X, deben extraerse las tripletas a partir de sus ontologías de dominio y de las anotaciones RDF basadas en esta ontología. Cada parte de la tripleta corresponde a una unidad de entrada para el modelo neuronal.

El modelo de clasificación neuronal propuesto es un Perceptrón Multicapa (MLP, sigla de su traducción al inglés Multilayer Perceptron). Éste tiene entradas correspondientes a cada componente de las tripletas. Cada componente de la tripleta consiste de dos palabras. Por lo tanto, son consideradas 6 unidades de entrada para el modelo MLP. El número de unidades ocultas está determinado por un procedimiento de prueba y error. Hay tantas neuronas de salida en el modelo como nodos a ser asociados. La activación de una neurona de salida consiste en producir un valor de 1 cuando la tripleta de entrada existe en el nodo correspondiente, y 0 en otros casos.

4.3.3. Fases de entrenamiento y correspondencia.

Durante el entrenamiento supervisado, se muestran al modelo los pares:

$PatrónEntrada = <Objeto;Atributo;Valor>$

con su correspondiente salida deseada, indicando a qué nodo o dominio Dx, Dy o Dz pertenece esta entrada:

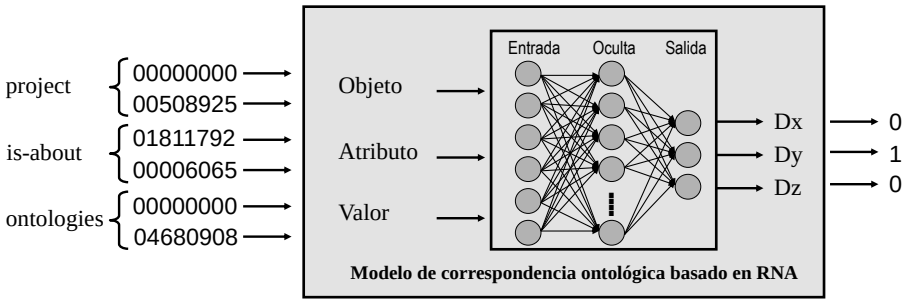

Figura 4.2: Modelo para OM propuesto basado en RNA y un ejemplo de patrón de entrenamiento.

PatrónDeseado = <Dx;Dy;Dz>.

Con el objeto de entrenar el modelo de OM basado en RNA, deben crearse estos ejemplos de entrenamiento. Cada ejemplo debe decirle a la red que cierta tripleta puede ser encontrada en cierto nodo. Por ejemplo, consideremos las ontologías y anotaciones que se muestra en la Figura 4.1. Un patrón de entrenamiento que indica que la tripleta *<project;is-about;ontologies>* puede encontrarse en la ontología del nodo B, pero no en A ni en C, debe tener la siguiente forma:

PatrónEntrada = <project;is-about;ontologies>
PatrónDeseado = <0;1;0>

Esto significa que, dado que hay proyectos en el nodo B que tienen interés de investigación en ontologías, su tripleta correspondiente sería *<project;is-about;ontologies>* y su salida deseada correspondiente sería *<0;1;0>*. Solamente el segundo valor del vector, que representa al nodo B, es igual a 1, indicando así que esta tripleta puede ser encontrada en el nodo B.

Una vez que son creadas las tripletas para cada nodo, deben ser convertidas de strings a números. Por esto, en este estudio se propone codificar los términos de las tripletas utilizando la base de datos Wordnet[4] , en la cual el nombre de un elemento es asociado con un código numérico llamado *synset offset*. Este código se representa mediante un número entero decimal de 8 dígitos. De esta manera, es posible alcanzar un esquema apropiado de codificación de patrones porque todos los términos pueden codificarse con un código de longitud fija. Sin embargo, muchos de los términos representados en WordNet son palabras simples. Por ejemplo, en el Wordnet 1.6 en Inglés, el término *Semantic Web* no es un sólo término. Esto es un problema que se soluciona asumiendo estos casos como dos

[4] http://wordnet.princeton.edu/

palabras independientes. Luego, un término de una tripleta puede representarse como un par de códigos, cuyos valores variarán dependiendo de si el término tiene:

i) *una palabra simple:* el código del término será formado por el código *synset* asociado con la palabra, incluyendo un código de 8 números cero en la primer posición, representando así la ausencia de otra palabra;

ii) *dos palabras:* el código del término será formado por la composición del código *synset* asociado a cada palabra.

La Figura 4.2 muestra también la codificación de una tripleta de ejemplo. Puede observarse que para *project* el código está formado por la combiación de cero y *00508925*: *<00000000;00508925>*. El código relacionado a *is-about* está formado por el código de *is 01811792* y el código de *about 00006065*:*<01811792;00006065>*. La codificación de una palabra simple es aplicada a la palabra *ontologies*.

Resumiendo, el patrón de entrenamiento sería: $PatrónEntrada = << 00000000; 00508925 >;< 01811792; 00006065 >;< 00000000; 04680908 >>$, PatronDeseado $= < 0; 1; 0 >$.

Del uso de la base de datos Wordnet para la codificación de los términos de una tripleta, surge un hecho interesante relacionado al uso de ontologías de diferentes lenguajes. Hay una traducción automática de tripletas utilizando el esquema de codificación de Wordnet para entrenar el modelo RNA, ya que un término en una tripleta tiene el mismo código en el Wordnet en Inglés y en Español. Por ejemplo, el término en Inglés *project* y su traducción al Español *proyecto* poseen el mismo código: *00508925*. Utilizando esta codificación única, es posible consultar a todos los nodos sin tener en cuenta el idioma de cada nodo. Conclusiones similares pueden obtenerse en el caso de los sinónimos. Por lo tanto, el idioma no es una barrera para esta propuesta de OM.

4.4. Aplicación al dominio de I+D.

Para entrenar el modelo de OM basado en RNA, no sólo se requieren ontologías, sino también anotaciones basadas en RDF. En consecuencia, se ha decidido llevar a cabo una evaluación específica en un dominio de aplicación, lo que tiene

dos ventajas principales: (i) es más realista que los bancos de prueba artificiales[5] , (ii) provee información muy específica. Esta evaluación fue dividida en tres fases: (1) se construyó un conjunto de datos para la evaluación, (2) el modelo basado en RNA del agente KSD fue entrenado y validado, y (3) se realizaron diferentes consultas para probar el modelo basado en RNA.

4.4.1. Datos y experimentos

El ejemplo de aplicación usado en este capítulo está basado en el Proyecto PAE-CELTIC[6] . Este proyecto involucró seis grupos de investigación, cuatro instituciones académicas, seis empresas y dos organizaciones no gubernamentales. Estas entidades están localizadas en diferentes ciudades de dos provincias diferentes en Argentina. Además, más de cien profesionales trabajan en el proyecto en diferentes áreas de Tecnología de la Información y Comunicaciones. No todos los participantes del proyecto están familiarizados con todas las áreas de investigación de los demás. Más aún, la estructura del proyecto puede cambiar en el tiempo y pueden agregarse nuevas entidades, lo que significa tener nuevas personas involucradas en él con nuevas áreas de trabajo (dominios) a las cuales consultar.

Con el objetivo de obtener un conjunto de anotaciones basadas en RDF para entrenar un clasificador RNA, fueron seleccionados los sitios web de cuatro grupos de investigación, involucrados en el proyecto PAE-CELTIC, y anotados semánticamente utilizando tres ontologías diferentes: KA, SWRC y OLID (previamente descriptas en la Figura 4.1). La Tabla 4.1 muestra los sitios web seleccionados y la ontología que fue utilizada para anotarlos. Nótese que en el caso de los nodos B y C, el idioma del sitio web y el de la ontología no es el mismo. Este hecho ciertamente puede hacer más difícil el proceso de asociación cuando los diferentes idiomas se mezclan en las anotaciones.

Utilizando el lenguaje de consulta RDQL[7] , fue obtenido un conjunto de tripletas para cada ontología y de anotaciones semánticas para cada nodo. Este conjunto fue preprocesado eliminando las tripletas vacías e incompletas, tales como las anotaciones que tienen al menos un elemento vacío, porque no proveen

[5] http://oaei.ontologymatching.org/
[6] http://www.celtic.santafe-conicet.gov.ar/pae
[7] http://www.w3.org/Submission/RDQL/

Tabla 4.1: Nodos y ontologías de dominio utilizadas para anotarlos.

Nodo	*Sitio Web*	*Idioma*	*Ontología*
A:CIDISI	http://cidisi.frsf.utn.edu.ar	Inglés	KA
B:INGAR	http://www.ingar.santafe-conicet.gov.ar	Inglés	SWRC
C:SINC	http://fich.unl.edu.ar/sinc/	Inglés	KA
D:CIMEC	http://www.cimec.org.ar	Español	OLID

la información útil. El conjunto final de tripletas tiene 159 elementos en total, compuesto de 53 anotaciones para cada dominio.

Los parámetros del modelo MLP fueron configurados de acuerdo a valores típicos. Los pesos y umbrales fueron inicializados aleatoriamente en el intervalo [1;-1] en el inicio de la fase de entrenamiento. Cada neurona de la capa oculta fue asociada con la función sigmoidea. El algoritmo de entrenamiento utilizado fue *Backpropagation*. Los datos de entrenamiento fueron normalizados a la función de activación de las neuronas ocultas, antes de entrar al modelo, dado que esto mejora significativamente el tiempo de entrenamiento y la precisión de la OM. El número de neuronas de entrada para el modelo MLP fue fijado en 6, considerando el código doble para cada término de la tripleta, tal como se indicó anteriormente. El número de neuronas en la capa oculta fue configurado empíricamente, de acuerdo a los datos de entrenamiento y la precisión deseada para el asociador. La salida posee una neurona especializada en el modelo para cada nodo, siendo en este caso 4 neuronas en la capa de salida, dado que hay cuatro nodos. Los valores permitidos para cada neurona de salida fueron 1 o 0, dependiendo de si la neurona reconoce o no un elemento perteneciente al nodo que representa.

Para el entrenamiento fue utilizado el método de validación cruzada con k particiones (Haykin, 2005). El conjunto de datos completo fue aleatoriamente dividido en $k = 3$ subconjuntos de igual tamaño, repitiendo los experimentos tres veces en cada partición. La estimación de la precisión total del modelo fue calculada mediante un promedio simple de las medidas de precisión sobre los tres conjuntos de prueba. Por cada nodo, hay 53 tripletas, 43 de las cuales se utilizaron para entrenamiento y 10 para validación.

Para el aprendizaje del modelo fueron utilizadas dos estrategias:

1. *Entrenamiento de orden aleatorio*: las tripletas de entrenamiento (pertenecientes y no pertenecientes a la clase) son presentadas, durante el entrenamiento, en un orden aleatorio al modelo.

2. *Primero uno conocido (known), luego tres desconocidos (unknown) (1 kn, 3 unkn)*: el modelo neuronal recibe las tripletas de entrenamiento pertenecientes a cada clase, en el siguiente orden: una tripleta perteneciente al nodo y tres que no pertenecen al mismo.

El modelo de OM basado en RNA ha sido comparado con el algoritmo H-Match (Castano y otros, 2006), un algoritmo muy reconocido para encontrar correspondencias entre ontologías pobladas mediante la evaluación de la afinidad semántica entre dos conceptos y considerar tanto su afinidad lingüística como conceptual. Para usar este algoritmo, se presenta a cada ontología una consulta de prueba (una palabra). Se utiliza el algoritmo H-Match para determinar si la ontología poblada tiene conceptos que se corresponden con la consulta.

El algoritmo H-Match provee un valor de afinidad semántica ($s_{t_i,n}$) para cada término de la tripleta $< t_i >$ comparado con cada ontología de dominio o nodo n. Los valores $s_{t_i,n}$ se combinan para obtener una medida de correspondencia promedio para la tripleta completa $< t_1, t_2, t_3 >$ contra la ontología del nodo, de acuerdo a la siguiente fórmula:

$$Av_{(<t_1,t_2,t_3>,n)} = \frac{s_{t_1,n} + s_{t_2,n} + s_{t_3,n}}{3} \tag{4.2}$$

Para determinar si la tripleta puede ser indicada como perteneciente o relacionada al nodo analizado n, la medida de afinidad semántica $Av_{(<t_1,t_2,t_3>,n)}$ tiene que ser más alta que el umbral α configurado empíricamente. Si éste es satisfecho, la tripleta $< t_1, t_2, t_3 >$ es considerada como que se corresponde a la ontología del nodo n.

Para el modelo de correspondencias basado en RNA propuesto, la evaluación del conjunto de prueba debe ser definido correctamente porque una tripleta de consulta $t_i =< t_1, t_2, t_3 >$ puede estar incluída en más de una ontología de nodo. Es decir, en este problema de clasificación hay muchos patrones que pertenecen a varias clases. Se determinó la salida para cada caso de prueba teniendo en cuenta el mínimo valor obtenido para cada neurona de salida durante el entrenamiento con el correspondiente conjunto de entrenamiento para el nodo n. Esto

Tabla 4.2: Tasas de reconocimiento (RR) para el modelo de OM basado en RNA sobre el conjunto de datos de prueba.

Dominios	entrenamiento *aleatorio*	entrenamiento *1 kn, 3 unkn*
RR_A	80,00 %	83,33 %
RR_B	76,67 %	76,67 %
RR_C	83,33 %	86,67 %
RR_D	76,67 %	63,33 %
$RR_{global(promedio)}$	79,17 %	77,50 %

significa que, durante el entrenamiento, fueron identificados mínimos umbrales de activación para cada neurona de salida que representa a una clase. Cuando el patrón de prueba fue presentado al modelo MLP, la salida del modelo fue evaluada considerando las neuronas que presentan una salida con un valor superior a su correspondiente umbral. De esta manera, más de una neurona (clase) puede ser considerada como respuesta a la tripleta de consulta de prueba.

4.4.2. Resultados y discusión

La Tabla 4.2 presenta los resultados obtenidos para la tasa de clasificación o reconocimiento (RR_i) del modelo basado en RNA para cada clase $i = A, \cdots, D$, como el promedio sobre el conjunto de prueba (10 tripletas para cada clase) para todas las k particiones ($k = 3$). Los resultados muestran que, en promedio, el modelo propuesto puede alcanzar altas tasas de clasificación de casi el 80 % en cada clase y en promedio, considerando los resultados sobre todas las clases (dominios).

Para corroborar la influencia del orden de los patrones de entrenamiento sobre la capacidad de clasificación del modelo o la tasa de reconocimiento, fueron utilizadas dos estrategias: entrenamiento *aleatorio* versus entrenamiento *1 kn, 3 unkn*. Como se muestra en la Tabla 4.2 hay una diferencia importante entre el esquema de entrenamiento aleatorio y el ordenado. Prestando atención a las tasas de clasificación para cada dominio en detalle, puede verse que la peor tasa alcanzada sobre el conjunto de prueba obtenido para el nodo D es, sin embargo, más alta que el 60 %. Esta tasa alcanza un máximo de aproximadamente 90 % para el nodo C cuando el clasificador fue entrenado con patrones ordenados para

Tabla 4.3: Ejemplos de consultas de OM

Tripletas de consulta	Nodo de Asociación: anotación original
1) <*image Processing, type, research Topic*>	C:<Image Processing; type; Research Topic>,
2) <*process Control, type, theme*>	C:<process Control; type; theme>
3) <*grupo, trabaja, objeto Relacional*>	B:<Paradigma Objeto Relacional; type; Publication>
4) <*persona, trabaja, objeto Relacional*>	B:<Paradigma Objeto Relacional; type; Publication>
5) <*project, is About, ontologies*>	A:<project,is-about,ontologies>
6) <*fellow, interest, Semantic Web*>	A:<fellow,interest,semanticWeb>
7) <*projecto, sobre, ontologías*>	A:<projecto, sobre, ontologías>
8)<*becario, interés, Web Semántica*>	A:<becario, interés, Web Semántica>

esta clase. La estrategia de entrenamiento aleatorio provee los resultados más altos en promedio para todas las clases, de casi el 80 %.

Con el objetivo de comparar la performance del modelo de OM basado en RNA y el algoritmo H-Match, se utilizaron ocho tripletas de consulta para prueba sobre 3 dominios, las que son presentadas en la primer columna de la Tabla 4.3. Las tripletas 1) y 2) existen en el nodo C. Las tripletas 3) y 4) son similares, teniendo solamente una diferencia en el primer término. Ambas están en Español y pueden encontrarse en el nodo B. Es necesario mencionar que todas estas tripletas están expresadas en el mismo idioma que el nodo que debería ser indicado como uno de los que contiene a la tripleta de consulta o uno similar en la correspondiente ontología anotada. Las tripletas 5) y 8) han sido seleccionadas para probar la influencia del idioma en el modelo de OM. Las tripletas 5) y 7) tienen exactamente los mismos términos pero expresados en Inglés y Español, respectivamente. Lo mismo ocurre con las tripletas 6) y 8). En la segunda columna de la Tabla 4.3 se indica el nodo correspondiente que contiene la tripleta de consulta o una similar.

La Tabla 4.4 muestra, en la primer columna, el número de la tripleta de consulta considerada en la prueba; a partir de la segunda columna hasta la sexta, se presentan los resultados de OM para los nodos anotados A,B y C; y la última columna muestra la salida global del modelo: la ontología del nodo que es indicado como el que contiene los términos utilizados en la consulta. Un nodo es considerado si un valor de OM es más alto que un umbral de 0,6. Cuando más de un nodo satisface este requerimiento, la salida del modelo es presentada como un ranking de potenciales nodos ordenados desde valor de correspondencia más alto al más bajo. Lo valores numéricos que dan soporte a cada salida del modelo están resaltados en la tabla utilizando diferentes formatos: negrita para el modelo RNA e itálica para el modelo H-Match. Los resultados del modelo RNA para

Tabla 4.4: Resultados de la OM para un modelo basado en RNA (entrenamiento aleatorio) vs. el algoritmo H-Match ($\alpha = 0{,}6$) en un conjunto de prueba.

	Nodo A		Nodo B		Nodo C		Salida del modelo	
Consulta	RNA	H-Match	RNA	H-Match	RNA	H-Match	RNA	H-Match
1)	$(A, 0{,}023)$	$(A, \mathit{0.733})$	$(B, 0{,}161)$	$(B, \mathit{0.702})$	$(C, \mathbf{0.750})$	$(C, \mathit{0.733})$	C	C, A, B
2)	$(A, 0{,}000)$	$(A, \mathit{0.677})$	$(B, 0{,}000)$	$(B, \mathit{0.682})$	$(C, \mathbf{1.000})$	$(C, \mathit{0.677})$	C	B, C, A
3)	$(A, 0{,}037)$	$(A, \mathit{0.643})$	$(B, \mathbf{0.987})$	$(B, 0{,}597)$	$(C, 0{,}000)$	$(C, \mathit{0.643})$	B	A, C
4)	$(A, 0{,}029)$	$(A, \mathit{0.679})$	$(B, \mathbf{0.990})$	$(B, \mathit{0.637})$	$(C, 0{,}000)$	$(C, \mathit{0.679})$	B	A, C, B
5)	$(A, \mathbf{0.750})$	$(A, \mathit{0.905})$	$(B, 0{,}000)$	$(B, \mathit{0.826})$	$(C, 0{,}000)$	$(C, \mathit{0.905})$	A	A, C, B
6)	$(A, \mathbf{0.750})$	$(A, \mathit{0.640})$	$(B, 0{,}000)$	$(B, \mathit{0.604})$	$(C, 0{,}000)$	$(C, \mathit{0.640})$	A	A, C, B
7)	$(A, \mathbf{0.750})$	$(A, \mathit{0.835})$	$(B, 0{,}000)$	$(B, \mathit{0.790})$	$(C, 0{,}000)$	$(C, \mathit{0.835})$	A	A, C, B
8)	$(A, \mathbf{0.750})$	$(A, \mathit{0.616})$	$(B, 0{,}000)$	$(B, 0{,}587)$	$(C, 0{,}000)$	$(C, \mathit{0.616})$	A	A, C

esta prueba corresponden a la fase de entrenamiento utilizando una estrategia de muestreo aleatorio, el que ha permitido obtener tasas de clasificación más altas para este problema, como se muestra en la Tabla 4.2. Para obtener los resultados del modelo H-Match, fueron evaluados los ejemplos de prueba nodo a nodo, configurando los parámetros del algoritmo de forma estándar como sigue: modelo de correspondencia = *intensivo*, correspondencia = *uno-a-uno*, adopta herencia = *falso*, estrategia de contexto vacío = *pesimista*, estrategia de correspondencia = *estándar (asimétrica)* y peso de la afinidad lingüística = *1,0*.

El análisis de estos resultados indica que el modelo RNA hace corresponder a todas las consultas de prueba; esto significa que en el 100 % de los casos de prueba, este modelo indica correctamente la ontología de dominio a la cual pertenece la tripleta de prueba. Para el algortimo H-Match, son correctamente asociadas siete de las 8 consultas de prueba (87,5 %). Es importante resaltar que el modelo neuronal siempre indica solamente un nodo. Mientras que, en la mayoría de los casos, la salida del modelo H-Match indica más de un nodo posible para responder a una consulta. Un análisis detallado ha sido realizado evaluando cada consulta de prueba. En relación al modelo H-Match, las tripletas 1), 5), 6), 7) y 8) son consideradas correctas porque el primer nodo indicado es el correcto. Para la segunda y cuarta tripleta, la salida es aceptada como correcta a pesar de que el nodo correcto no aparece en el primer lugar. En la tripleta de prueba 3) la salida es considerada incorrecta porque el nodo que debería haber sido indicado como el poseedor de una respuesta no aparece siquiera en el ranking. Debe decirse que, debido a que el algoritmo H-Match otorga más importancia a la estructura de la ontología que a sus instancias, este algoritmo ha proporcionado siempre la misma respuesta para los nodos anotados con la misma ontología.

Volviendo al escenario motivador para este estudio, se debe recordar que el

modelo de OM es un componente del agente KSD, el que recibe consultas transportadas por agentes móviles en búsqueda de respuestas a las mismas. Cuando una lista de nodos son indicados como potenciales respuestas a la consulta, todos esos nodos deberían ser visitados en el orden estipulado, lo que hace más probable una pérdida de tiempo y una sobrecarga de la red. Esto es definitivamente una ventaja importante del modelo RNA por sobre el modelo H-Match porque el primero siempre indica sólo un nodo.

En resumen, se puede afirmar que, para todos los casos de prueba, el modelo de OM basado en RNA ha proporcionado resultados satisfactorios. A juzgar por los resultados obtenidos, el modelo propuesto es más preciso a la hora de elegir los nodos potenciales que responden a una consulta, en comparación con un algoritmo clásico de correspondencia entre ontologías. Nótese que con respecto a las tripletas 5) a 8), éstas corresponden a traducciones de las mismas palabras, y el modelo basado en RNA provee la misma respuesta correcta para todos los casos. Esta es una característica importante del modelo propuesto, porque la combinación del uso de un esquema inusual de codificación de palabras (tal como Wordnet) y un modelo de OM basado en redes neuronales ha demostrado ser independiente del idioma. Esta es una ventaja importante cuando el idioma de consulta es diferente del lenguaje de las anotaciones de una ontología de dominio.

4.5. Resumen

Este capítulo ha presentado un modelo clasificador neuronal para la correspondencia ontológica. El modelo de correspondencia ontológica basado en redes neuronales ha sido explicado en detalle, basado en un modelo Perceptrón Multicapa. Además, se propuso la codificación de las entradas al modelo neuronal mediante el uso del corpus Wordnet. Se aplicó el modelo propuesto en el dominio de I+D con anotaciones semánticas de cuatro ontologías anotadas.

Los resultados obtenidos fueron analizados y discutidos no sólo en cuanto a la capacidad del modelo neuronal de aprender correctamente la información que contiene cada dominio y ontología anotada, sino que también se realizó la comparación del modelo propuesto con un algoritmo de búsqueda de correspondencias entre ontologías reconocido en la literatura, llamado H-Match.

Como consecuencia del estudio explicado en este capítulo, se ha logrado

comunicar los resultados a un congreso internacional con referato (Rubiolo y otros, 2009) y publicar también en una revista indexada con Factor de Impacto 2012 de 3.643 (Rubiolo y otros, 2012).

CAPÍTULO 5

Redes neuronales para modelar relaciones en bioinformática

5.1. Introducción

El presente capítulo propone un nuevo enfoque para la minería de relaciones entre perfiles de expresión de genes utilizando un conjunto de redes neuronales artificiales (RNA). En primer lugar, se realizará una introducción a las redes de regulación de genes y su relación con las series temporales (Sección 5.2). Luego, se explicará en detalle como se entrena un modelo RNA utilizando los datos temporales de los perfiles de expresión para el modelado de las relaciones gen a gen, aplicando este enfoque propuesto al descubrimiento de la red de regulación de genes (GRN, del inglés Gene Regulatory Network) subyacente en un conjunto de datos de perfiles de expresión de genes (Sección 5.3). Para validar la metodología de descubrimiento de relaciones propuesta, se utilizará un conjunto de datos artificiales con interacciones conocidas previamente, con el objeto de confirmar la precisión y la sensitividad de la propuesta, y luego se aplicará el enfoque a un conjunto de datos biológicos reales pertenecientes a la especie *Solanum lycopersicum* (Sección 5.4). Los resultados obtenidos han sido enviados a una revista indexada con Factor de Impacto 2012 de 1.616 (Rubiolo y otros, 2013a) y se encuentran en proceso de revisión.

5.2. Series temporales y redes de regulación de genes

Una red de regulación de genes es un asociación abstracta de las regulaciones entre genes que ocurren en las células vivas, la que puede ayudar a predecir el comportamiento del sistema de tales organismos. Durante los últimos años, se han propuesto diferentes enfoques para descifrar la complejidad de la regulación entre genes (Hecker y otros, 2009).

Los genes poseen una actividad medible mediante observaciones en un número determinado de instantes en el tiempo, constituyendo así una serie temporal que se conoce como perfil de expresión de genes. Actualmente, un tema de gran interés en boinformática es la reconstrucción de una red regulatoria de genes a partir de esas series temporales, descubriendo así la red de relaciones gen a gen existente (Hartemink, 2005). En otras palabras, el objetivo es determinar un patrón de activaciones e inhibiciones entre genes que conforman la red de regulación subyacente en los datos.

Dados los niveles de expresión de un conjunto de genes candidatos, medidos en diferentes instantes de tiempo, se pueden desarrollar métodos formales para modelar la interacción entre ellos (Wang y otros, 2007). De hecho, en los últimos años se ha estudiado este problema mediante metodologías basadas en datos (Jong, 2002; Muraro y otros, 2013; Nounou y otros, 2012; Pinna y otros, 2013; Styczynski y Stephanopoulos, 2005). Estas metodologías son capaces de procesar de manera flexible grandes cantidades de datos y obtener resultados a partir del análisis de los mismos. Por ejemplo, las Redes Booleanas propuestas para este problema solamente consideran la expresión o no expresión de un gen, sin tener en cuenta el nivel de expresión existente, teniendo así una resolución inadecuada de la dinámica del problema (Bornholdt, 2008; Higa y otros, 2013). Las redes Bayesianas son representadas como grafos con la distribución de probabilidad conjunta de los genes (Werhli y Husmeier, 2007). Este modelo puede capturar efectivamente el ruido inherente en los datos y el comportamiento estocástico en la expresión de genes, pero su alto costo computacional obstaculiza la aplicación del modelo a redes con gran cantidad de genes. Se han propuesto redes bayesianas dinámicas que extienden a las redes bayesianas anteriores para incluir

comportamiento temporal (Husmeier, 2003; Ibrahim y otros, 2011). También, los algoritmos genéticos han sido aplicados para predecir las vías de regulación representadas como una matriz de influencia (Ando y Iba, 2001; Mitra y otros, 2011). Es posible aplicar este enfoque a problemas con poca cantidad de datos, el cual puede optimizar simultáneamente una gran cantidad de parámetros, y resolver modelos no lineales. Más recientemente han aparecido propuestas basadas en modelos ocultos de Markov (Ram y Chetty, 2011), máquinas de soporte vectorial (Ao y Palade, 2011) y redes neuronales (Liu y otros, 2011; Yang y otros, 2013). Particularmente, en (Knott y otros, 2010) fue presentado un modelo basado en redes neuronales para la identificación de redes de regulación de genes a partir de los datos temporales de expresión de genes, en el que fueron utilizados tanto redes feed-forward como redes de Elman (de tipo recurrente). Los modelos propuestos tratan de imitar estructuralmente a las redes genéticas, y poseen la capacidad de asociar la intensidad de las interacciones entre genes con los pesos de la red. Un conjunto de datos artificiales que contiene una GRN subyacente previamente conocida, fue utilizado para probar este enfoque, y han sido encontradas algunas de las interacciones existentes, si bien la red completa original no puedo ser reconstruida. Recientemente, los resultados de un estudio comparativo han destacado al enfoque basado en redes neuronales como el método de mejor desempeño de entre las propuestas recientes (Hache y otros, 2009).

Las redes neuronales artificiales pueden utilizarse para modelar las redes de genes, observando la actividad de un par de genes, en un determinado número de instantes de tiempo. Así, el modelado de la regulación entre genes se convierte en un proceso altamente combinatorio, en el que deben ser analizadas todas las combinaciones posibles entre genes a fin de descubrir sus relaciones. El uso de RNA para este modelado requiere entrenarlas para poder predecir la regulación de un gen a partir del perfil de un gen candidato a posible regulador. Ajustando los pesos sinápticos, las RNA modifican su configuración para modelar cada conexión entre genes, resultando en un error mínimo en la predicción del perfil del gen regulado.

En cualquier GRN, las interacciones complejas que existen entre los genes pueden ser tanto instantáneas como retardadas en el tiempo (Chowdhury y otros, 2013). Por esto, los perfiles de expresión de genes pueden considerarse como series temporales (Barker y otros, 2011). De hecho, estas variables bien podrían no ser

perfiles de expresión de genes sino estar asociadas al comportamiento de cualquier elemento o componente de un sistema dinámico. Este capítulo propone un enfoque novedoso para el descubrimiento de GRN a partir de los perfiles temporales de expresión de genes, utilizando un conjunto de redes neuronales del tipo MLP con retardos temporales en la entrada. Los perfiles de expresión de genes son utilizados para entrenar un conjunto de redes neuronales. Cada RNA es diseñada para descubrir, para un perfil de gen deseado en la salida, el potencial regulador en la entrada, modelando su interación gen a gen durante un período de tiempo. La capacidad de cada RNA entrenada para modelar las posibles relaciones es evaluada mediante el error de generalización del modelo luego del proceso de entrenamiento. Para cada posible relación encontrada, se computa una matriz de puntuación para detectar las más probables. Se proponen, además, tres reglas para la minería de GRN a partir de la matriz de puntuación, lo que permite descubrir cada correcta relación gen a gen entre todas las posibles. La propuesta se explica en detalle mediante un ejemplo en la sección siguiente.

5.3. Minería de relaciones entre genes con redes neuronales

Esta sección explica cómo una RNA puede entrenarse utilizando datos temporales de perfiles de expresión de genes para modelar relaciones gen a gen. Luego, se presenta el nuevo enfoque propuesto para la construcción de una GRN a partir de las relaciones descubiertas.

Los perfiles de expresión de genes representan la actividad de los genes observada sobre un número de instantes de tiempo. Por ejemplo, en la Figura 5.1 se muestra un segmento del comportamiento en el tiempo de dos perfiles de genes, A y B, posiblemente ambos relacionados. Puede verse que, para el gen A, en $t = 4$, la señal comienza a incrementarse, mientras que para el gen B este comportamiento comienza a ser percibido en $t = 6$. Puede decirse entonces que el gen A tiene cierta influencia sobre la variable B, y el retardo temporal entre el momento de activación de estos dos genes es aproximadamente de dos instantes de tiempo. Estas series de datos temporales pueden utilizarse para entrenar una red neuronal con el objetivo de modelar esta relación gen a gen, que es posiblemente

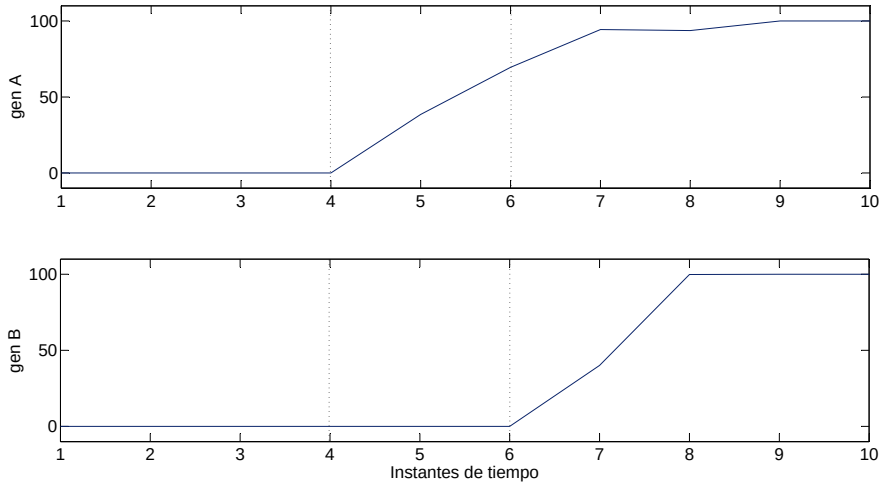

Figura 5.1: Identificación del retardo temporal entre dos series de datos temporales.

parte de una red de relaciones entre genes.

Como ya se ha dicho, es posible utilizar una red neuronal para modelar la relación entre dos señales. De acuerdo a la capacidad de modelado dentro de un determinado número de épocas de entrenamiento, podría ser posible detectar una regulación entre genes representados por dos series temporales. En el marco teórico (Capítulo 2, Sección 2.2.3) se introdujo el modelo MLP con retardos temporales en la entrada, el que puede considerar explícitamente la estructura temporal de una señal. Lo específico de este tipo de red particular, como ya se mencionó anteriormente, es que el número de entradas está vinculado al número de retardos temporales considerados en la señal de entrada, representados en la ventana τ, tal como puede observarse en la Figura 5.2. Es decir, habrá τ+1 entradas, considerando los valores de la señal de entrada para los tiempos t a $t-\tau$. El valor del retardo temporal τ considerado en la señal de entrada puede determinarse automáticamente analizando cada relación gen a gen. De esta manera, todas las posibles combinaciones de relaciones entre genes deberían ser analizadas para determinar el mínimo retardo temporal existente entre pares de relaciones en el conjunto de datos.

Esta propuesta consiste de tres pasos: 1) modelar todas las posibles relaciones gen a gen utilizando un conjunto de RNA, 2) dar una puntuación a cada

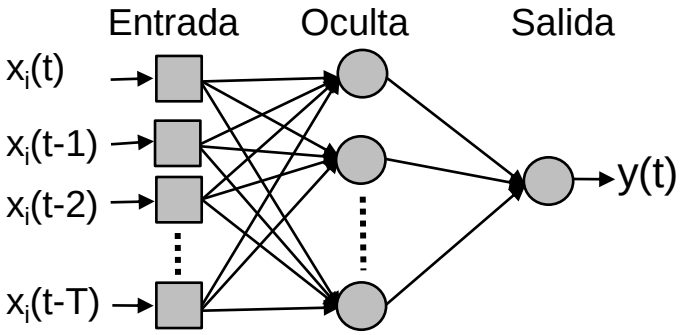

Figura 5.2: Modelo RNA con retardos temporales en la entrada.

relación encontrada, 3) aplicar las reglas para descubrir la GRN subyacente en los datos. Estos pasos serán explicados en detalle en las siguientes subsecciones.

5.3.1. Redes neuronales para modelado de relaciones entre variables temporales

Modelar las relaciones gen a gen implica construir un modelo RNA para cada posible combinación de genes *regulador-regulado* y entrenar este modelo con las series temporales correspondientes. Es importante encontrar un modelo que tenga en cuenta adecuadamente la dinámica de los datos. Para este propósito, se pueden tener en cuenta valores bajos, medios y altos para la cantidad de épocas de entrenamiento. Si luego de un gran número de iteraciones el error de entenamiento es alto, es posible concluir que no hay relación entre las señales analizadas; mientras que si el error es bajo, esto puede interpretarse como un indicativo de que tal relación existe. Sin embargo, si la relación entre dos señales no es los suficientemente clara, independientemente del número de iteraciones que se utilice en el experimento, al final del proceso de entrenamiento el error será de todos modos alto. Similarmente, si la relación entre dos genes es lo suficientemente evidente, cualquiera sea el número de iteraciones, el error será bajo. Por lo tanto, para hacer frente a este problema, prefiere seleccionarse un número bajo de iteraciones.

Los experimentos deben repetirse varias veces con el objeto de alcanzar resultados estadísticamente más robustos. Por lo tanto, habrá un conjunto (o "pool") de modelos MLP especializados en cada relación. Globalmente, esto puede ser considerado como un conjunto de redes neuronales cuyo objetivo es medir cuán reproducible es la relación entre dos series temporales. Es decir, cuán modelable

es la relación gen a gen que pertenece a una GRN.

La capacidad de una red neuronal entrenada de modelar una red de genes es evaluada utilizando el MSE, el cual es el promedio de las diferencias entre las predicciones de la actividad de los genes, realizada mediante una RNA, y el verdadero perfil de expresión de los mismas, sobre todas las medidas. Por lo tanto, para cada entrenamiento realizado en el conjunto de modelos, se mide el error de generalización mediante el procedimiento de validación cruzada, a fin de encontrar las diferencias y similitudes entre los genes de entrada y salida deseada. Este error será considerado en el próximo paso para aplicar el método de puntuación, explicado en la próxima subsección.

5.3.2. Puntuación de las relaciones

Con el objetivo de determinar qué gen es el potencial regulador de otro, es necesario otorgar un puntaje a cada relación modelada. Este puntaje está basado en cuántas veces ha sido alcanzado el menor error, considerando todas las repeticiones de cada experimento. Por ejemplo, supongamos que tenemos cinco genes $\{A, B, C, D, E\}$. Se listan todos los candidatos a reguladores del gen B. Luego, se otorga un puntaje a cada posible relación *regulador-regulado*: $\{A \rightarrow B, C \rightarrow B, D \rightarrow B, E \rightarrow B\}$. Esto se lleva a cabo para todos los genes, de manera tal que se evalúan todas las alternativas *regulador-regulado* posibles.

Los resultados experimentales deben ordernarse de manera tal de que el menor valor del error esté en el tope de la lista y el mayor, al final, como puede observarse en la Tabla 5.1. Con el objeto de mostrar más claramente el ejemplo, en esta tabla se muestran solamente las primeras cinco repeticiones (primera columna) y los dos primeros pares con mínimo valor MSE para cada repetición. En el ejemplo, el experimento que evalúa si el gen A regula al gen B posee el mínimo valor de MSE en todas las repeticiones. Luego de esto, se aplica el método de puntuación analizando el ranking resultante, considerando las dos alternativas siguientes: puntuación por puntos (PS, sigla de su traducción al inglés Point Scoring) y puntuación por ranking (RS, sigla de su traducción al inglés Ranking Scoring).

La primer alternativa, la puntuación PS, consiste en otorgar 2 puntos a la relación regulador-regulado con el menor error (la primera posición en la lista), y 1

Tabla 5.1: Ejemplo del ranking de error luego de cinco repeticiones de un experimento que tiene por objetivo descubrir el regulador del gen B.

Repetición	***Regulador***	***Regulado***	***Error***
1	A	B	$5{,}21e^{-10}$
	C	B	$6{,}10e^{-10}$
2	A	B	$8{,}92e^{-10}$
	D	B	$9{,}57e^{-10}$
3	A	B	$0{,}35e^{-09}$
	E	B	$1{,}75e^{-09}$
4	A	B	$2{,}19e^{-09}$
	D	B	$8{,}32e^{-09}$
5	A	B	$0{,}98e^{-08}$
	C	B	$1{,}64e^{-08}$

punto a la relación siguiente con el menor error (la segunda posición en la lista). Este procedimiento de puntuación es repetido tantas veces como repeticiones fueron hechas. Por ejemplo, de la lista que se muestra en la Tabla 5.1 es posible ver que se han realizado cinco repeticiones para determinar cuál es el regulador del gen B. Estas repeticiones están separadas en la lista utilizando líneas de puntos. En todas ellas, la relación $A \rightarrow B$ obtiene el primer puesto, por lo que su puntaje es 10. Las relaciones $C \rightarrow B$ y $D \rightarrow B$ obtienen el segundo puesto en dos repeticiones, por lo tanto se le asigna 2 puntos a cada una. Por último, el puntaje obtenido por $E \rightarrow B$ es 1 ya que obtiene el segundo puesto solamente en una repetición.

Una vez que los puntos han sido asignados, es posible construir una matriz de puntuación para representar cada posible relación gen a gen, como puede observarse en la Figura 5.3. Las filas representan los genes potenciales reguladores mientras que las columnas muestran a los genes regulados. Los elementos de la diagonal no son tenidos en cuenta debido a que la autoregulación no es considerada en este estudio. El relleno de la matriz de puntuación consiste en colocar los puntajes en la celda de la matriz correspondiente a la relación *regu-*

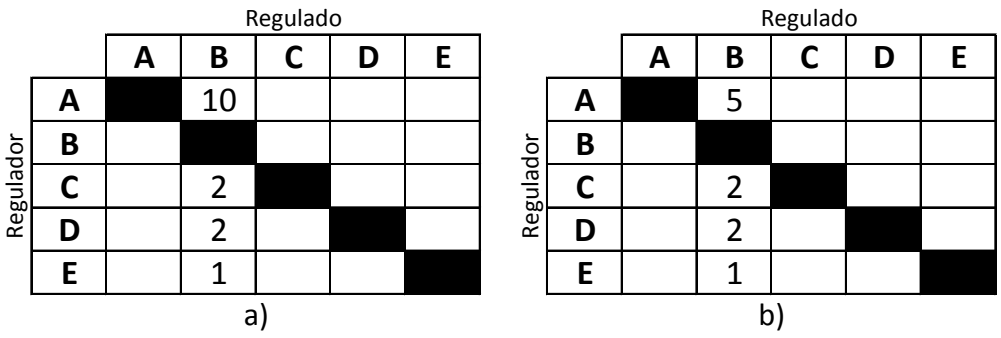

Figura 5.3: Ejemplo de la matriz de puntaje para minería de GRN. a) Matriz de puntaje por puntos (PS). b) Matriz de puntaje por ranking (RS). Se muestran solamente los valores relacionados al gen B según el ejemplo motivador.

lador−regulado. Por ejemplo, la segunda columna de la matriz que se muestra en la Figura 5.3a) se llena con los puntajes calculados anteriormente mediante la aplicación del criterio de puntuación PS sobre la lista para la regulación del gen B.

La alternativa de puntuación RS tiene en cuenta las primeras N relaciones de la lista, y cada relación obtiene 1 punto tantas veces como esté presente en las primeras N posiciones de la lista. Por ejemplo, considerando los 10 primeros experimentos listados en la Tabla 5.1, es posible observar que la relación $A \rightarrow B$ está presente 5 veces, $A \rightarrow C$ y $A \rightarrow D$ se muetran 2 veces, y $A \rightarrow E$ solamente 1 vez. Por lo tanto, sus puntajes serán 5, 2, 2 y 1, respectivamente. Estos resultados de la aplicación del método RS se muestran en la Figura 5.3b). El mismo procedimiento puede aplicarse para completar el resto de la tabla, analizando todos los experimentos y repeticiones realizados para todos los genes estudiados.

5.3.3. Reglas para la reconstrucción de las relaciones

Como se dijo anteriormente, durante el proceso de puntuación, la matriz se completa con los puntajes obtenidos luego del análisis de los resultados del conjunto de RNA definidas en la subsección 5.3.1. En la Figura 5.4a) se presenta un ejemplo de una matriz de puntuación completa según el criterio PS. Si se observa con detalle, es posible identificar que hay algunas relaciones dudosas

o difíciles de discernir presentes en la misma. El objetivo de las reglas que se presentan en esta subsección es resolver estos conflictos.

Regla de la Simetría

El primer caso dudoso puede ocurrir cuando, entre dos genes particulares, uno de ellos actúa como gen regulador en una relación y como gen regulado en otra. Por ejemplo, en la Figura 5.4a) puede observase cómo la regulación $D \rightarrow C$ tiene 2 puntos y la regulación $C \rightarrow D$ posee 5 puntos. Luego de la aplicación de esta regla, la que se muestra en la tabla de la Figura 5.4b), se mantiene solamente la relación que posee el máximo valor entre ambas. Dos casos análogos están presente entre las relaciones $D \rightarrow B$ y $B \rightarrow D$, y entre las relaciones $E \rightarrow B$ y $B \rightarrow E$.

Debe notarse que la regla de la *Simetría* puede ser aplicada considerando el número de puntos y/o el error de la relación bajo estudio. En otras palabras, la decisión de mantener una relación u otra puede realizarse teniendo en cuenta solamente el puntaje, solamente el error, o una combinación de ambas alternativas. Tener en cuenta toda esta información en la decisión permite a la regla evaluar cada caso individual consistentemente, previniendo la calificación como correcta de una relación que en realidad es errónea. Por esto se propone, asumiendo por ejemplo que es necesario determinar el ganador entre dos posibles regulaciones: $A \rightarrow B$ y $B \rightarrow A$, utilizar una medida φ como:

$$\varphi = \alpha\rho + (1 - \alpha)\varepsilon, \tag{5.1}$$

donde α es un parámetro de pesado y $\alpha \, \epsilon \, [0, 1]$. El valor ε relaciona a los *errores* de ambas regulaciones como sigue:

$$\varepsilon = \frac{e_{BA} - e_{AB}}{\text{máx}\{e_{AB}, e_{BA}\}}, \tag{5.2}$$

donde e_{AB} es el error del modelo neuronal que reproduce $A \rightarrow B$, mientras que e_{BA} es el error del modelo inverso $B \rightarrow A$. Cuando $\varepsilon > 0$, $A \rightarrow B$ es el modelo que posee error más bajo. Por el contrario, cuando $\varepsilon < 0$ el modelo $B \rightarrow A$ posee el menor valor. Análogamente, ρ relaciona a ambos *puntajes* de los modelos como sigue:

$$\rho = \frac{s_{AB} - s_{BA}}{\text{máx}\{s_{AB}, s_{BA}\}}, \tag{5.3}$$

donde s_{AB} es el puntaje del modelo $A \rightarrow B$, mientras que s_{BA} es el puntaje obtenido por el modelo inverso $B \rightarrow A$, por lo que $\rho > 0$ si el primer modelo tiene el máximo puntaje.

El parámetro α es utilizado para evaluar las diferencias entre errores y puntajes para tomar una decisión sobre una relación regulador-regulado. Es decir, α es considerado como un peso que permite seleccionar entre dos alternativas o la combinación de ambas, y su valor puede modificarse de acuerdo a si el puntaje, o el error, es más relevante, con $\alpha \epsilon [0, 1]$. Aplicando (5.1) al ejemplo anterior, cuando $\varphi > 0$ puede inferirse que $A \rightarrow B$ es la relación ganadora.

Regla de desencadenado

En la Figura 5.4b) puede observarse otro caso especial entre B, D y E. Analizando la matriz, la relación $B \rightarrow D \rightarrow E$ está presente, pero existe también la relación $B \rightarrow E$. En este caso, llamado *regulación en cadena*, es necesario seleccionar una de las relaciones o caminos posibles y descartar la otra alternativa. Esta regla indica que debe seleccionarse la regulación en cadena con más cantidad de genes ya que la más corta está de alguna manera incluída en la más larga. De esta manera, se remueve de la matriz la relación $B \rightarrow E$, resultando la matriz en la Figura 5.4c).

Regla del Umbral

El último caso especial se presenta cuando, para un gen regulado, hay un conjunto de posibles reguladores con bajo valor PS, compitiendo con un candidato cuyo valor de PS es más grande. Por ejemplo, en la Figura 5.4c) los posibles reguladores de B son A con 10 puntos y C con 2 puntos. En este caso es necesario descartar aquellos candidatos con un número de puntos que esté por debajo de un umbral. En la aplicación de esta regla es posible también considerar tanto el número de puntos como el error cometido en el modelo. Por lo tanto, esta regla es tenida en cuenta para excluir aquellas relaciones cuyos errores y puntajes estén por debajo de un umbral determinado $\theta \epsilon [0,1]$. Para poder aplicar esta regla, en primer lugar, los valores de los puntajes y de los errores son normalizados en la escala [0,1]. En segundo lugar, se aplica el parámetro θ para calcular un valor del umbral para el puntaje $\theta_s = \theta$ y uno para el error $\theta_e = (1 - \theta)$. Luego, el umbral excluirá aquellas relaciones cuyos puntajes estén por debajo de θ_s y errores por arriba de θ_e.

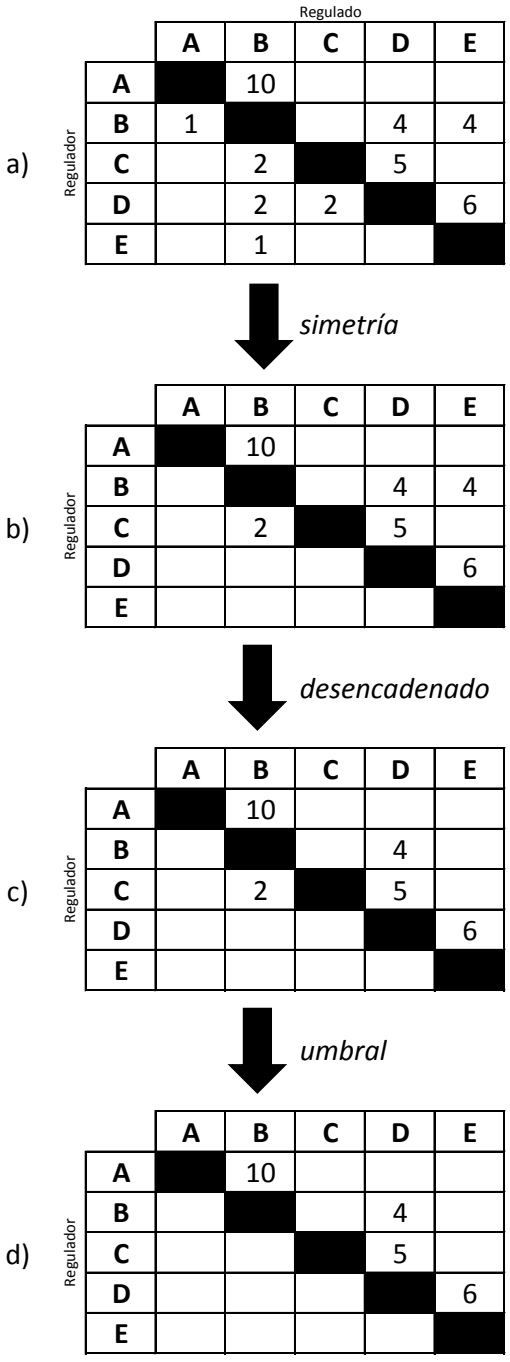

Figura 5.4: Aplicación de las reglas a partir de la matriz PS. a) Matriz de puntuación *Original*. b) Matriz resultante luego de la aplicación de la regla *Simetría*. c) Matriz resultante luego de la aplicación de la regla *Desencadenado*. d) Matriz resultante luego de la aplicación de la regla *Umbral*.

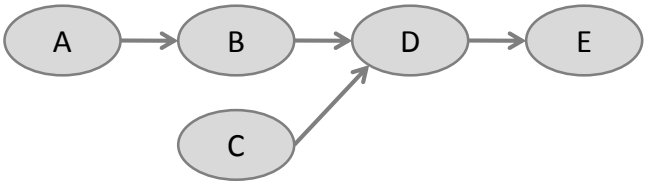

Figura 5.5: Una simple GRN descubierta como resultado de la aplicación de las reglas de minería.

Finalmente, luego de la aplicación de las reglas explicadas anteriormente, puede obtenerse una GRN a partir de la matriz de puntuación resultante. Por ejemplo, a partir de la matriz en la Figura 5.4d) es posible obtener la GRN que se muestra en la Figura 5.5, dibujando las relaciones *regulador−regulado* presentes en la matriz.

5.4. Aplicación: descubriendo redes de regulación reales entre genes

En esta sección se describe un conjunto de datos artificiales con conexiones conocidas sobre las cuales se probó la propuesta de este capítulo. Además se presenta un conjunto de datos biológicos obtenido de los perfiles de expresión de los genes de la vía metabólica de la vitamina E en el tomate (Quadrana y otros, 2013), el cual ha sido utilizado para validar también la propuesta con un caso de estudio real. Luego, se muestran y analizan los resultados experimentales.

5.4.1. Datos y experimentos

Datos artificiales

El simulador de datos artificiales utilizado para probar esta propuesta fue desarrollado por (Smith y otros, 2002) y mejorado por (Yu y otros, 2004). Estos trabajos modelaron una vía metabólica de regulación de genes de una estructura de red arbitraria y midieron los valores de expresión de genes en instantes de tiempo discretos. El método fue el mismo que se utilizó en (Knott y otros, 2010) para modelar los valores de expresión de genes y validar el modelo de regulación

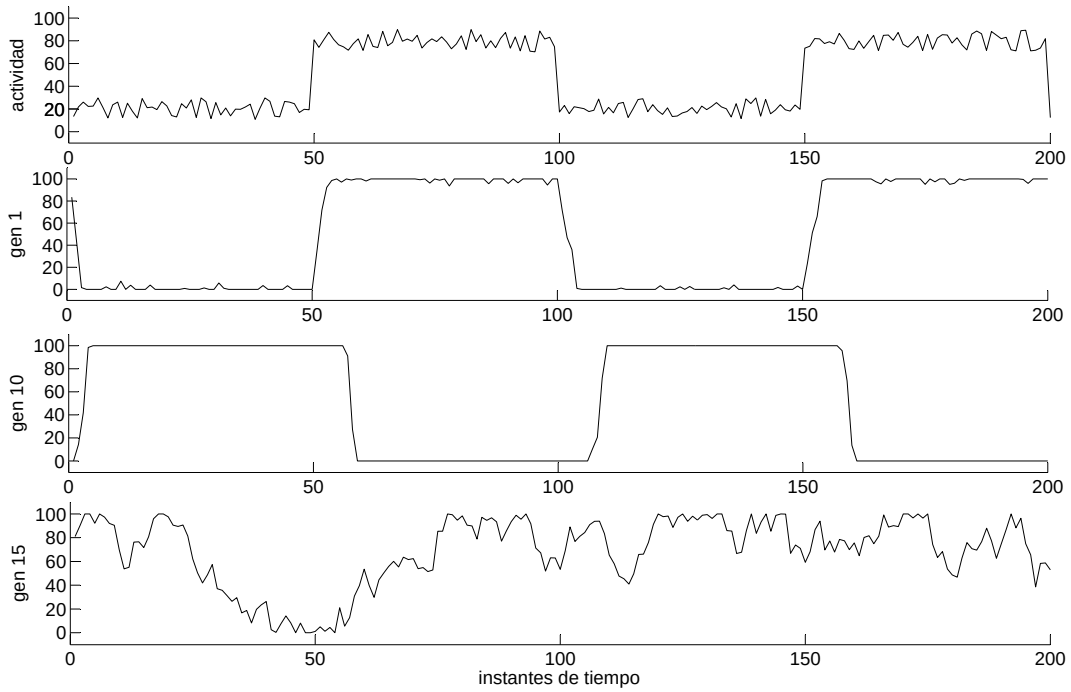

Figura 5.6: Serie temporal para la actividad, y la expresión de los genes 1, 10 y 15.

de genes, contra el cual se hará la comparación de esta propuesta.

La Figura 5.6 muestra ejemplos de algunas de las series temporales utilizadas en el estudio. Puede observarse que un incremento en la *actividad* (trazo superior) es seguido por una regulación positiva del *gen 1* (segundo trazo desde arriba) con un leve retardo temporal. Cuando la actividad cae, el *gen 1* vuelve a bajar cerca de su nivel mínimo, también con un leve retardo temporal. El *gen 10* (tercer trazo desde arriba) es regulado en forma negativa, considerablemente más tarde que la respuesta del *gen 1* a la actividad, mostrando que los efectos regulatorios se propagan a través de la red en el tiempo. El *gen 15* (trazo inferior) es un gen que no está regulado y puede verse claramente como va cambiando aleatoriamente de nivel en un amplio rango, y sin relación con la actividad o los otros genes mostrados en la gráfica. Para estas series temporales, la red conocida correspondiente, surgida del simulador antes mencionado, se muestra en la Figura 5.7. Ésta contiene 20 genes, de los cuales solamente 10 están asociados entre si a través de interacciones regulatorias positivas o negativas, configuradas con un valor de +0,20 (Smith y otros, 2002), mientras que los 10 genes restantes sirven

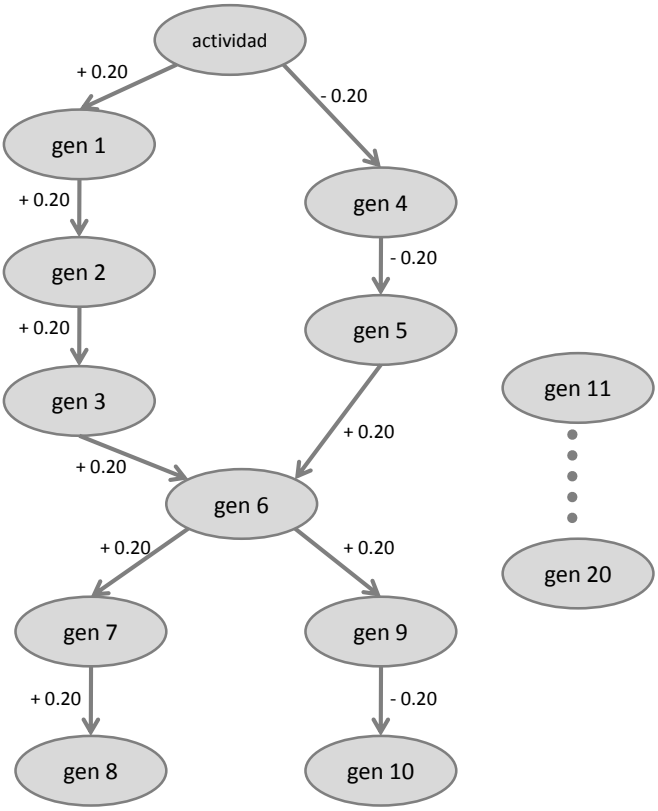

Figura 5.7: Topología de la GRN conocida, presente en el conjunto de datos artificiales.

como ruido. Los niveles de expresión de dos de los genes en la red están directamente afectados por el nivel de la actividad. Estos dos genes, a su vez, afectan el nivel de expresión de 8 genes que están "río abajo" en el flujo regulatorio.

En el simulador, en cada instante de tiempo, el nivel de expresión de los genes en la red están gobernados por los niveles de expresión de sus reguladores, un factor de degradación y un factor de ruido. Por el contrario, los niveles de expresión de los genes restantes fluctúa o se atenúa aleatoriamente dentro de los límites inferior y superior del nivel de expresión. Debido a la naturaleza estocástica del ruido en el sistema, la salida diferirá levemente en cada réplica pero será gobernada mediante las mismas interacciones de la red subyacente. En este trabajo, se utilizaron 10 réplicas, cada una de las cuales contiene 20 niveles de expresión de genes en 200 instantes de tiempo, muestreados en intervalos de cinco minutos. El 80 % del conjunto de datos fue utilizado para el entrenamiento, mientras que el 20 % restante fue utilizado para la prueba.

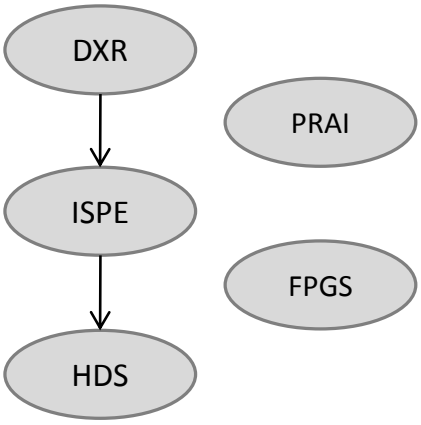

Figura 5.8: Topología de parte de una GRN biológica: VTE en tomate.

Datos reales

Por su parte, el conjunto de datos biológicos reales contiene datos sobre el tomate cultivado (*Solanum lycopersicum*), el cual es uno de los vegetales mundialmente más consumidos y algunos estudios han sugerido que es una importante fuente de vitamina E (VTE). La Figura 5.8 muestra parte de la vía metabólica propuesta en (Quadrana y otros, 2013) luego de la identificación experimental de una colección de genes de tomate involucrados en la biosíntesis de la VTE. En ese trabajo, el análisis de los datos de expresión reveló algunos genes clave altamente asociados con el contenido de VTE en las hojas y frutos del tomate, los que aparecen como candidatos a ser manipulados para mejorar el valor nutricional del fruto del tomate. La expresión relativa de los genes en diferentes tejidos del tomate fue obtenida de 4-6 plantas y agrupadas en tres réplicas biológicamente independientes. Las medidas obtenidas representan los valores medios de las réplicas, normalizados contra la muestra de la expresión relativa más baja. Los perfiles de expresión de los genes fueron medidos en diferentes etapas del crecimiento del tomate: (i) la hoja comienza a desarrollarse y consume alimento proveniente de otra fuente hasta que puede producirlo ella misma (sink leaves), (ii) la hoja puede producir alimentos por si misma (source leaves), (iii) el fruto está verde (green), (iv) el fruto se convierte en verde maduro (mature green), (v) inicio o comienzo de la transición (breaker), y (vi) el fruto está maduro (ripe). Para los experimentos que tienen por objeto encontrar parte de la GRN biológica real, se han seleccionado los siguientes genes que codifican enzimas: 2-C-methyl-D-erythritol 4-phosphate synthase (DXR); 4-(cytidine

50-diphospho)-2-C-methyl-D-erythritol kinase (ISPE); 4-hydroxy-3-methylbut-2-enyl-diphosphatesynthase (HDS); phosphoribosylanthranilate isomerase (PRAI); folylpolyglutamate synthase (FPGS).

En cuanto a la configuración de los experimentos, en este estudio fueron considerados dos valores: $\tau = \{2,3\}$. En cuanto a la capa oculta, se adoptó una simple heurística en la que el número de neuronas ocultas sean el 50 %, 100 %, 150 % y 200 % del número de entradas. Por ejemplo, para una capa de entrada con $\tau = 2$ neuronas, la capa oculta puede tener 2, 4, 6 u 8 neuronas. La capa de salida tiene solamente una neurona, la que representa el valor de la expresión del gen regulado en el tiempo t.

Los pesos y umbrales fueron inicializados de acuerdo al algoritmo de inicialización Nguyen-Widrow (Nguyen y Widrow, 1990), que escoge valores a fin de distribuir la región activa de cada neurona aproximadamente de manera uniforme en todo el espacio de entrada. Los modelos fueron entrenados utilizando una función que actualiza los pesos y los umbrales de acuerdo a la optimización Levenberg-Marquardt (Marquardt, 1963). El número de épocas de entrenamiento utilizadas como criterio de parada fueron 50, 100 y 500. Los parámetros α para la aplicación de la regla de la Simetría, como así también los parámetros θ para la regla Umbral, fueron probados entre 0 y 1, con un factor incremental de 0,05. Para medir el desempeño de los resultados, se han utilizado dos medidas: precisión (P) y sensitividad (S), previamente explicadas en el marco teórico (sección 2.4.2).

5.4.2. Resultados y discusión

En esta subsección se muestran los resultados experimentales obtenidos sobre dos problemas de minería de regulaciones entre genes. En primer lugar, se mostrarán los resultados que se obtuvieron en el conjunto de datos artificial. Se mostrará la GRN descubierta y se realizará una discusión acerca de la comparación de estos resultados con un trabajo relacionado. Por último, se mostrarán y analizarán los resultados de la aplicación de la propuesta de minería de GRN sobre un conjunto de datos biológico real.

Tabla 5.2: Datos Artificiales. Valores de precisión (P) para $\alpha = 0{,}5$, $\theta_s = 0{,}75$, y $\theta_e = 0{,}05$, cuando se alcanzan los valores más altos de sensitividad (S) (100 %) en cada experimento.

Tamaño de ventana (τ)	Épocas	P [%]
2	50	**99,12**
	100	95,91
	500	96,49
3	50	97,95
	100	93,27
	500	97,95

La Tabla 5.2 presenta resultados globales del enfoque de minería propuesto sobre el conjunto de datos artificiales. La primer columna indica el tamaño de la ventana utilizada en los experimentos, representando el retardo temporal tenido en cuenta en la entrada de la RNA. En la segunda columna se muestran las diferentes épocas de entrenamiento consideradas. En la tercer columna se presenta el valor de precisión. Para todos los casos la sensitividad fue del 100 %.

Analizando esta tabla, pueden resaltarse dos resultados importantes. El primero es que, utilizando el enfoque propuesto en este capítulo, es posible obtener todas las relaciones existentes de la GRN conocida, la que ha sido utilizada como referencia. Esto es posible ya que la sensitividad obtenida fue del 100 % en todos los casos. En segundo lugar, los valores de precisión están entre 93 % y 99 % por lo que se han descubierto muy pocas relaciones extra o falsos positivos, es decir, que no existen en la red real. El mejor valor de precisión obtenido fue de 99,12 % (se muestra en negrita en la tabla).

Un análisis diferente puede realizarse mirando los valores de sensitividad correspondientes al valor más alto de precisión obtenido en cada experimento, como se muestra en la Tabla 5.3. En este caso, los valores de sensitividad están entre 88,89 % y 100 %, por lo que también, en el mejor de los casos, la red de referencia es obtenida efectivamente. Esto significa que son obtenidas todas las relaciones gen a gen que deben ser descubiertas, y se descubren otras pocas relaciones que no están presentes en la red de control. En la Tabla 5.3, el valor de precisión más alto alcanzado es del 99,12 % y corresponde a un valor de sensitividad del 100,00 %, lo que significa que el enfoque propuesto es capaz de descubrir una GRN que es muy similar a la red de referencia. Se pueden obtener conclusiones

Tabla 5.3: Datos artificiales. Valores de sensitividad (S) cuando se alcanza el valor más alto de precisión (P) en cada experimento, para $\alpha = 0{,}5$, $\theta_s = 0{,}75$, y $\theta_e = 0{,}05$.

Tamaño de ventana (τ)	Épocas	P [%]	S [%]
2	50	**99,12**	**100,00**
	100	98,89	88,89
	500	97,95	88,89
3	50	**97,95**	**100,00**
	100	97,95	88,89
	500	**97,95**	**100,00**

similares de las filas 4 y 6 de la Tabla 5.3, en la que los valores de sensitividad también alcanzan el 100 %, correspondiente a un 97.95 % de precisión.

Analizando ambas Tablas 5.2 y 5.3, los valores de sensitividad y precisión muestran que con esta propuesta es posible encontrar una GRN mediante el uso de RNAs con retardos temporales en la entrada. Otro hecho importante que surge de la consideración de ambas tablas es que este enfoque permite al usuario obtener diferentes soluciones. Éste, puede preferir soluciones que descubren todas las relaciones (verdaderos positivos) en el conjunto de datos pero con el agregado de algunas relaciones que no existen realmente (falsos positivos), por un lado. Éstas se reflejan con los valores más altos de sensitividad en la Tabla 5.2. Por otro lado, el usuario puede preferir soluciones que descubran la menor cantidad posible de relaciones falsas (falsos positivos), aún cuando esta situación implique una pérdida de relaciones deseadas (verdaderos positivos). Éstas se reflejan en los valores más altos de precisión en la Tabla 5.3. En otras palabras, el usuario puede decidir entre obtener todas las relaciones correctas aceptando que algunas relaciones incorrectas aparezcan ($S = 1$ pero $P < 1$), o descubrir solamente las relaciones correctas aún si no se puede obtener algunas de las relaciones correctas, debido a la rigidez del método ($P = 1$), pero asegurando que las relaciones encontradas sean verdaderas o correctas.

Por ultimo, a partir de ambas tablas es posible también resaltar que no hay diferencias relevantes entre los resultados de los experimentos considerando los dos tamaños de ventana diferentes. Los resultados de ambos análisis fueron consistentes para todas las opciones de los parámetros.

Para este conjunto de datos artificiales, tras la aplicación de las reglas de

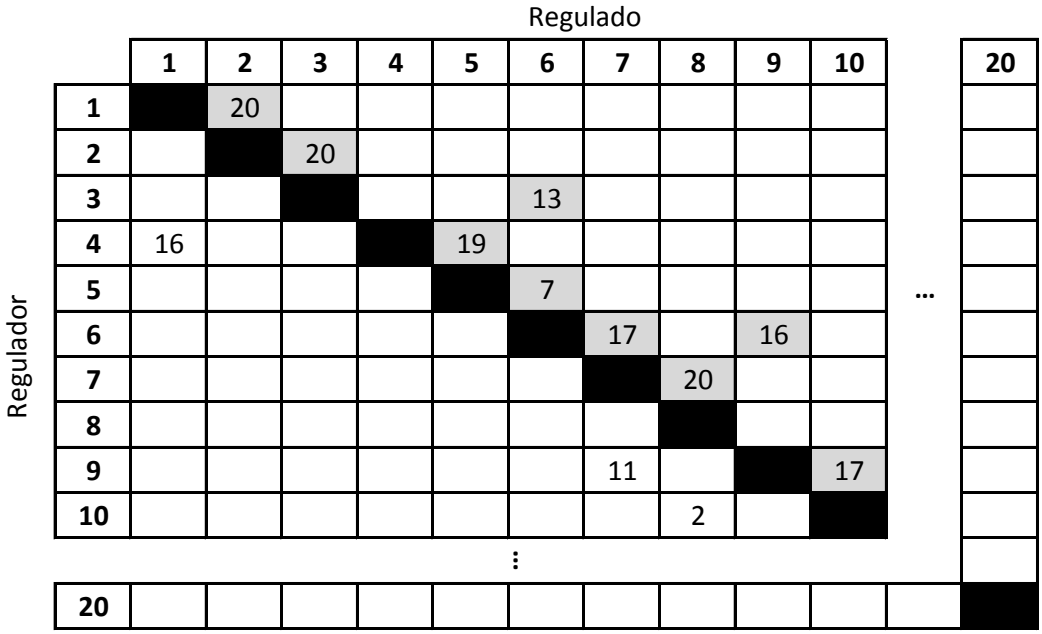

Figura 5.9: Datos artificiales. Matriz de puntuación resultante luego de la aplicación de las reglas de minería sobre el conjunto de datos artificiales. Las filas y columnas representan genes. Las celdas negras son relaciones de autorregulación (no consideradas). Las celdas en gris son relaciones presentes en la GRN de referencia.

minería, se utiliza la matriz resultante para construir una GRN. Como fue explicado anteriormente, las relaciones gen a gen se dibujan a partir del contenido de esta matriz. La Figura 5.9 muestra la matriz resultante correspondiente al mejor caso presentado en las Tablas 5.2 y 5.3 (primer fila en ambas tablas). Como se dijo anteriormente, la autoregulación no es objeto de este estudio, por lo que la diagonal de la matriz no se considera. Las celdas en gris indican las relaciones gen a gen de la red de referencia. Los números en las celdas son los puntajes obtenidos por cada relación. Para simplificar la figura, las celdas entre las filas y columnas 11 y 19 fueron escondidas ya que no se han descubierto relaciones entre esos genes durante el proceso de minería. De la matriz de puntuación en la Figura 5.9 es posible construir la GRN que se muestra en la Figura 5.10. Puede verse

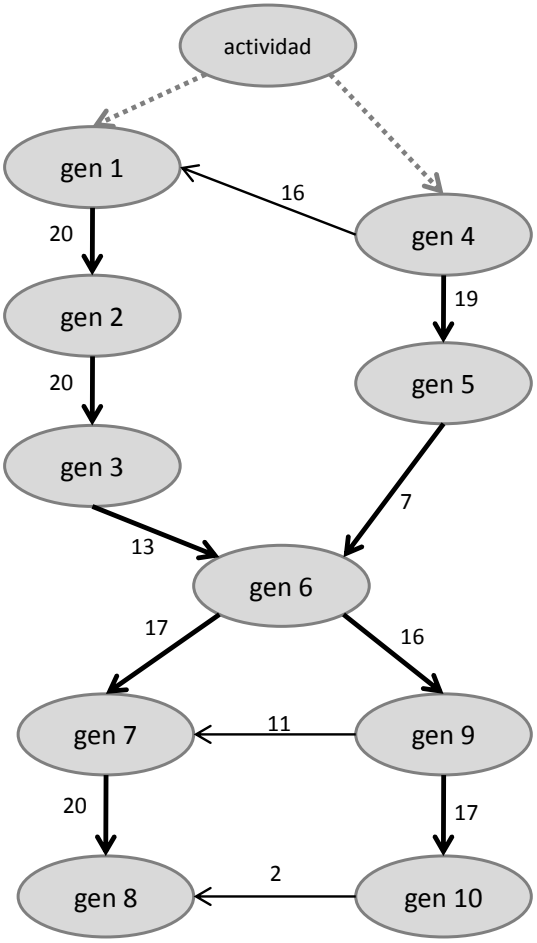

Figura 5.10: Datos artificiales. GRN que fue descubierta. Las líneas punteadas indican la influencia de la actividad sobre los genes 1 y 4. Las líneas gruesas son las relaciones descubiertas por el enfoque de minería que corresponde con las relaciones de referencia en el conjunto de datos. Las líneas finas son las relaciones descubiertas que no están presentes en la red subyacente. Cada línea tiene un valor de puntuación correspondiente.

claramente que el enfoque de minería de relaciones entre genes presentado en este capítulo fue capaz de descubrir efectivamente todas las interacciones regulatorias existentes entre los genes, tal como se mostró en las líneas de trazo grueso. Se descubrieron tres relaciones extras (no presentes en la red de referencia), que están indicadas por líneas de trazo fino.

Finalmente, un hecho importante a discutir es la presencia de interacciones no deseadas entre genes. Como se dijo anteriormente, se encontraron también tres relaciones extra gen a gen. Analizando la magnitud y el signo de la influencia de estas relaciones entre genes en la GRN de referencia mostrada en la Figura 5.7, es posible ver que la red de regulación tiene una actividad inicial a partir de la cual

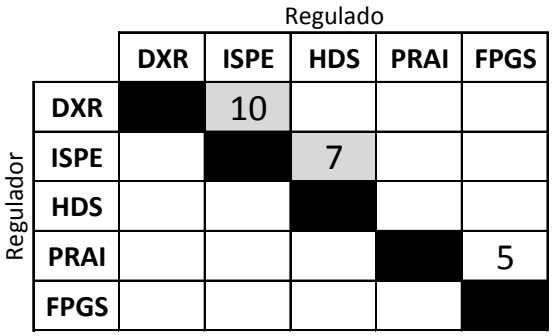

Figura 5.11: Matriz de puntuación resultante luego de la aplicación de las reglas de minería sobre un conjunto de datos biológico real. Las filas y columnas representan genes. Las celdas negras son relaciones de autorregulación (no consideradas). Las celdas en gris son relaciones presentes en la GRN biológica de referencia.

surge el nivel de expresión de los primeros genes. Por ejemplo, la actividad afecta directamente a los genes 1 y 4, en paralelo, pero con signo diferente. Luego, el gen 1 afecta positivamente al gen 2, mientras que el gen 4 regula negativamente al gen 5, y así sucesivamente. Esto siginifica que hay dos flujos de señales regulatorias provenientes de la misma actividad inicial, pero con diferente signo de influencia. Por lo tanto, es razonable que la propuesta de este capítulo encuentre una relación extra entre el gen 1 y el gen 4 porque, aunque indirectamente, existe, y aquellos genes dependen de la misma dinámica temporal inicial. Es posible obtener conclusiones similares considerando las regulaciones extras descubiertas 9→7 y 10→8, gobernadas en paralelo por el comportamiento del gen 6. Consecuentemente, puede concluirse que, sin cualquier conocimiento $a-priori$ de la red GRN subyacente, la propuesta es capaz de encontrar las regulaciones gen a gen que realmente existen, más algunas otras relaciones posiblemente redundantes entre genes cuando existe un camino en paralelo en la red GRN real.

Para este mismo problema, el método propuesto en (Knott y otros, 2010) fue capaz de inferir con éxito solamente ocho de las nueve interacciones existentes, siendo incapaz de identificar al gen 3 como uno de los reguladores de 6. Otro trabajo relacionado, que usó el mismo conjunto de datos y la misma GRN (Smith y otros, 2002), fue también incapaz de predecir la regulación del gen 6 por el gen 3. Además, encontró cinco conexiones inexistentes entre genes.

También se ha probado el enfoque propuesto con un conjunto de datos

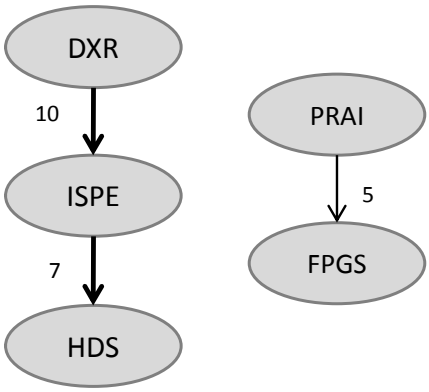

Figura 5.12: GRN biológica descubierta. Las líneas punteadas indican la influencia de la actividad sobre los genes 1 y 4. Las líneas gruesas son las relaciones descubiertas por el enfoque de minería que corresponde con las relaciones de referencia en el conjunto de datos. Las líneas finas son las relaciones descubiertas que no están presentes en la red subyacente. Cada línea tiene su valor de puntuación correspondiente.

biológico. La Figura 5.11 muestra la matriz resultante luego de la aplicación del enfoque de minería sobre un conjunto de datos biológico introducido previamente en la subsección 5.4.1, y en la Figura 5.12 la reconstrucción de la GRN correspondiente.

Es posible observar, a partir de los resultados, que las relaciones descubiertas son las regulaciones reales entre los genes VTE en el tomate reportados en (Quadrana y otros, 2013), por lo que la sensitividad alcanza, para este problema real, el 100 %. Sin embargo, una relación que no existe fue encontrada también, lo que resulta en una precisión del 95 % para este problema. De acuerdo a estos resultados, se puede afirmar que la propuesta es capaz de descubrir regulaciones existentes entre algunos de los genes de la VTE en el tomate, y así ser capaz de reconstruir su GRN. Más aún, a pesar de que ha sido encontrada una relación extra que no estaba presente en la realidad, el enfoque fue capaz de distinguir a los genes que realmente están relacionados directamente(DXR, ISPE, HDS) de aquellos que no (PRAI, FPGS). Esto puede ser muy útil para los biólogos ya que les permite focalizar su atención sobre este conjunto de genes que aparecen como relacionados. Es decir, mientras que generalmente ellos tienen que evaluar manualmente el conjunto total de los genes candidatos para construir una GRN, este enfoque de minería de relaciones entre genes puede automáticamente ofrecerles un subconjunto de genes y alguna estructura básica de la GRN sobre la

que puedan concentrar su estudio.

5.5. Resumen

En el presente capítulo, se explicó en detalle un enfoque novedoso para reconstruir una GRN a partir de los datos de expresión de varios genes candidatos. Este enfoque requirió del modelado de la interacción entre cada par de genes utilizando una RNA. Específicamente, se utilizó un conjunto de redes perceptrón multicapa para analizar todas las posibles combinaciones de relaciones gen a gen entre los genes presentes en el conjunto de datos. La capacidad de modelar efectivamente cada relación fue medida de acuerdo al error de generalización de cada modelo neuronal, obteniéndose así un ranking de los modelos con menor error cometido en el entrenamiento. Sobre todos estos posibles resultados se utilizaron dos métodos de puntuación para poder determinar las interacciones más probables en el conjunto de datos. A partir de la matriz de puntuación resultante, se aplicaron un conjunto de reglas de minería de relaciones para descubrir la GRN subyacente.

El método propuesto fue probado realizando varios experimentos sobre un conjunto de datos artificiales surgidos a partir de un simulador, que representan a una GRN previamente conocida. Además, se utilizó un conjunto de datos biológicos perteneciente a los perfiles de expresión de genes del tomate involucrados en la producción de Vitamina E. Los resultados, en ambos casos, mediante el cálculo de las medidas de desempeño de sensitividad y precisión, demostraron que la propuesta es capaz de descubrir efectivamente todas las relaciones gen a gen existentes y reconstruir la GRN subyacente en los datos. Esto ha sido enviado a una revista indexada con Factor de Impacto 2012 de 1.616 (Rubiolo y otros, 2013a) y al momento de la redacción de la presente tesis se encuentra en proceso de revisión.

CAPÍTULO 6

Conclusiones y Trabajo Futuro

Este capítulo muestra la conclusiones, principales contribuciones y trabajos futuros de esta tesis.

6.1. Principales Contribuciones

El presente trabajo de Tesis ha propuesto nuevos modelos, enfoques y algoritmos para minería de datos basados en redes neuronales artificiales, particularmente utilizando el ampliamente conocido modelo Perceptrón Multicapa. Todas estas propuestas han sido efectivamente probadas y validades utilizando bases de datos reconocidas en la literatura, como así también aplicadas a la resolución de problemas de gran interés en la actualidad, en áreas tales como la compresión de modelos, Biometría, Web Semántica y Bioinformática.

Se ha propuesto un método para obtener una representación compacta de un clasificador neuronal tipo MLP utilizando las salidas de diferente orden del modelo neuronal de Volterra. Se ha propuesto un algoritmo que permite la extracción de los parámetros de la red neuronal de Volterra de los parámetros de una MLP, luego de haber sido ésta entrenada. Un segundo algoritmo permite la obtención de diferentes salidas del modelo neuronal de Volterra cuando se recibe un nuevo punto de datos a ser clasificado. En primer lugar, se desarrolló un nuevo modelo para comprimir clasificadores neuronales simples llamados Volterra-NN. Éste ha sido probado para la compresión de un modelo simple en un conjunto de datos conocido en la literatura (Rubiolo y otros, 2010). Luego, se realizó una evaluación extendida del nuevo modelo propuesto para obtener una representación compacta de un arreglo de MLPs, proponiendo como consecuencia el modelo aV-NN. Se utilizaron dos diferentes bases de datos conocidas en la literatura, permitiendo demostrar así que es posible aplicar el modelo propuesto a problemas

de clasificación de creciente complejidad, y para diversas áreas de aplicación (Rubiolo, 2012). Por último, se ha utilizado para la compresión de un modelo más complejo, sobre un problema de reconocimiento de rostros, obteniendo casi la misma precisión que tres configuraciones diferentes de arreglos de clasificadores MLP (Rubiolo y otros, 2013b). En todos estos casos, los modelos clasificadores originales han sido significativamente reducidos a modelos que requieren menos parámetros. Los resultados experimentales han demostrado las capacidades del modelo V-NN propuesto para comprimir una solución con altas tasas de reconocimiento y ahorro de espacio. Además, se propuso una medida para selección de modelos de compromiso entre las soluciones posibles, la que permite considerar la prioridad de una de las tasas sobre la otra, de una forma simple. Esta medida fue útil para indicar a un modelo obtenido, entre las soluciones posibles, como el mejor de todos, considerando ambas tasas a la vez pero con diferentes pesos (Rubiolo y otros, 2013b).

Por otro lado, se ha presentado la efectividad del uso de un modelo clasificador neuronal para resolver el problema de la correspondencia entre ontologías. A diferencia de otras propuestas, este modelo está basado en una red neuronal que tiene en cuenta la información a nivel de esquema y a nivel de instancia para el entrenamiento de un clasificador para cada nodo anotado en la red. Además, el idioma en que se expresa la ontología podría ser diferente de la lengua en la que el sitio web está anotado. A través de los resultados obtenidos para un caso de estudio que involucra nodos pertenecientes al dominio de investigación y desarrollo, se puede afirmar que tal modelo es capaz de aprender a partir de varias etiquetas e instancias de ontologías heterogéneas. Un conjunto de tripletas de consulta han sido utilizadas para probar el modelo propuesto. En todos los casos, el modelo de correspondencia ontológica basado en un clasificador neuronal ha proporcionado una respuesta precisa. Los resultados experimentales han mostrado la efectividad y eficiencia del método propuesto, cuando se lo ha comparado con otro algoritmo de asociación clásico (Rubiolo y otros, 2012, 2009).

Además, se ha presentado una nueva propuesta de uso de redes neuronales para obtener las relaciones presentes entre datos de series temporales, en particular aplicado al descubrimiento o minería de redes de regulación de genes. Esta propuesta requirió utilizar un conjunto de modelos perceptrón multicapa para modelar cada interacción posible entre un par de genes y así analizar todas las

combinaciones posibles entre los genes presentes en el conjunto de datos. La capacidad de modelar efectivamente cada relación fue medida de acuerdo al error de generalización de cada modelo neuronal, obteniéndose así un ranking de los modelos con menor error cometido en el entrenamiento. Sobre todos estos posibles resultados se utilizaron dos métodos de puntuación para poder determinar las interacciones más probables en el conjunto de datos. A partir de la matriz de puntuación resultante, se aplicaron un conjunto de reglas de minería de relaciones para descubrir la GRN subyacente. Se han llevado a cabo un variedad de experimientos, tanto sobre un conjunto de datos artificial, el que simula los datos de perfiles de expresión de genes de una red de regulación conocida, como así también sobre un conjunto de datos biológicos perteneciente a los perfiles de expresión de genes del tomate involucrados en la producción de Vitamina E. Mediante el cálculo de las medidas de sensitividad y precisión, los resultados experimentales han demostrado que el enfoque propuesto es capaz de descubrir efectivamente todas las relaciones gen a gen presentes en los datos y reconstruir, de esta manera, la red de regulación de genes subyacente en los datos (Rubiolo y otros, 2013a).

6.2. Trabajo Futuro

Sin dudas, queda mucho trabajo futuro en las tres áreas de trabajo para la que se presentaron soluciones en el marco de esta tesis. No solamente en cuanto al aumento de la complejidad de los modelos, algoritmos y enfoques propuestos, sino también a la variedad de modelos neuronales que podrían ser utilizados para resolver estos mismos problemas y no fueron abordades en esta tesis.

En cuanto a la compresión de clasificadores neuronales utilizando el modelo V-NN, los trabajos futuros deberían implicar una aplicación de estos métodos a problemas de mayor complejidad, que involucren problemas reales de diversas áreas de aplicación, como así también un mayor número de clases. Además, las capacidades de compresión del modelo V-NN deberían ser probadas en ensambles de diferentes tipos de modelos neuronales, tales como redes recurrentes, parcialmente recurrentes y redes profundas (DNN, del inglés Deep Neural Networks). Sería realmente interesante estudiar estas diferentes topologías neuronales para poder establecer el impacto sobre las capacidades de compresión ofrecidas por

el modelo Volterra-NN propuesto. La compresion de redes DNN, las cuales son cada vez más utilizadas hoy en día en diversas aplicaciones, por ejemplo en reconocimiento de voz en teléfonos celulares, facilitaríaa su almacenamiento y uso en dispositivos con limitada capacidad de almacenamiento (Yu y otros, 2013). Las redes profundas son modelos probabilísticos generativos que se componen de múltiples capas de variables latentes estocásticas y que se entrenan principalmente con métodos no supervisados (Bengio, 2009). Estos métodos no supervisados, no lineales y jerárquicos, permiten realizar una reducción dimensional de los datos para proyectarlos en un espacio donde los clusters resulten más fácilmente identificables (Yu y Deng, 2011).

Por último, evaluando la propuesta de la aplicación de redes neuronales para el modelado de relaciones entre series temporales en bioinformática, sería interesante probar la propuesta en bases de datos de gran tamaño, y datos de perfiles de expresión de genes asociados a diversos procesos biológicos reales y completos. También sería interesante tener por objeto de estudio la determinación de la dinámica temporal de cada dato de expresión de genes, para poder determinar automáticamente el número de retardos temporales a ser considerados en la entrada de cada modelo neuronal. En coincidencia con las propuestas anteriores, debería estudiarse la inclusión de un ensamble de diferentes tipos de redes neuronales en los experimentos, de manera tal de que cada interacción particular gen a gen pueda ser modelada por el tipo de modelo neuronal más adecuado. Finalmente, y dado que cada perfil de expresión de gen no es sino una serie temporal, se podría analizar la utilización de este enfoque para el descubrimiento de relaciones variable a variable en problemas de otras áreas de aplicación que también involucren dinámica temporal en sus datos.

APÉNDICE A

Resultados suplementarios en reconocimiento de rostros

A.1. Base de datos de rostros ORL del AT&T Laboratories Cambridge

Tabla suplementaria 1 Tasa de reconocimiento (RR_i) para cada clase $i = 1, 2, .., 36$ para un clasificador aV-NN (a=36) y sus correspondientes salidas V-NN de primer $s^{(1)}$, segundo $s^{(2)}$ y tercer $s^{(3)}$ orden, para la base de datos de rostros ORL de AT&T Laboratories Cambridge. Los experimentos fueron realizados utilizando un modelo $36MLP_{40,40,1}$, entrenado con la información de 36 sujetos de la base de datos ORL.

	$36\mathrm{MLP}_{40,40,1}$									
$\mathrm{RR}_i[\%]$	RR_1	RR_2	RR_3	RR_4	RR_5	RR_6	RR_7	RR_8	RR_9	RR_{10}
$a\mathrm{MLP}_{NI,NH,NO}$	97,22	97,22	97,22	97,22	97,22	97,22	97,22	97,22	97,22	97,22
$36\text{V-NN}_{s^{(1)}}$	95,83	94,44	95,83	95,83	95,83	100,00	97,22	97,22	94,44	95,83
$36\text{V-NN}_{s^{(2)}}$	95,83	95,83	94,44	98,61	94,44	94,44	100,00	97,22	98,61	98,61
$36\text{V-NN}_{s^{(3)}}$	93,06	98,61	94,44	98,61	98,61	95,83	97,22	95,83	95,83	100,00
$\mathrm{RR}_i[\%]$	RR_{11}	RR_{12}	RR_{13}	RR_{14}	RR_{15}	RR_{16}	RR_{17}	RR_{18}	RR_{19}	RR_{20}
$a\mathrm{MLP}_{NI,NH,NO}$	97,22	90,28	97,22	97,22	97,22	97,22	97,22	97,22	97,22	88,89
$36\text{V-NN}_{s^{(1)}}$	95,83	94,44	95,83	94,44	94,44	95,83	98,61	97,22	94,44	95,83
$36\text{V-NN}_{s^{(2)}}$	95,83	95,83	93,06	97,22	95,83	94,44	95,83	97,22	93,06	95,83
$36\text{V-NN}_{s^{(3)}}$	95,83	97,22	93,06	95,83	95,83	98,61	98,61	95,83	94,44	95,83
$\mathrm{RR}_i[\%]$	RR_{21}	RR_{22}	RR_{23}	RR_{24}	RR_{25}	RR_{26}	RR_{27}	RR_{28}	RR_{29}	RR_{30}
$a\mathrm{MLP}_{NI,NH,NO}$	97,22	97,22	97,22	97,22	97,22	97,22	97,22	97,22	97,22	97,22
$36\text{V-NN}_{s^{(1)}}$	98,61	97,22	98,61	98,61	98,61	95,83	91,67	94,44	95,83	97,22
$36\text{V-NN}_{s^{(2)}}$	94,44	95,83	94,44	100,00	95,83	97,22	94,44	93,06	98,61	98,61
$36\text{V-NN}_{s^{(3)}}$	100,00	98,61	94,44	97,22	95,83	98,61	97,22	98,61	94,44	97,22
$\mathrm{RR}_i[\%]$	RR_{31}	RR_{32}	RR_{33}	RR_{34}	RR_{35}	RR_{36}				
$a\mathrm{MLP}_{NI,NH,NO}$	97,22	97,22	97,22	97,22	97,22	97,22				
$36\text{V-NN}_{s^{(1)}}$	98,61	97,22	95,83	98,61	95,83	94,44				
$36\text{V-NN}_{s^{(2)}}$	91,67	97,22	95,83	97,22	94,44	94,44				
$36\text{V-NN}_{s^{(3)}}$	95,83	93,06	94,44	95,83	95,83	93,06				

Tabla sumplementaria 2 Comparación de las tasas de reconocimiento global (RR) y de ahorro de espacio (SS) para un clasificador aV-NN (a=36) y sus correspondientes salidas V-NN de primer $s^{(1)}$, segundo $s^{(2)}$ y tercer $s^{(3)}$ orden, para la base de datos de rostros ORL de AT&T Laboratories Cambridge. Los experimentos fueron realizados utilizando un modelo $36\mathrm{MLP}_{40,40,1}$, representando 36 sujetos de la base de datos ORL.

		$36MLP_{40,40,1}$	
$P_{aMLP_{NI,NH,NO}} \rightarrow$		60516	
$\mathrm{RR}[\%] \rightarrow$		96,79	
3V-NN	$P_{as^{(i)}}$	RR[%]	SS[%]
$36\text{V-NN}_{s^{(1)}}$	1440	96,29	97,62
$36\text{V-NN}_{s^{(2)}}$	30960	95,99	48,84
$36\text{V-NN}_{s^{(3)}}$	88560	96,37	−46,34

A.2. Base de datos de rostros FERET Facial Recognition Technology

Tabla suplementaria 3 Tasa de reconocimiento (RR_i) para cada clase $i = 1, 2, .., 40$ para un clasificador aV-NN (a=40) y sus correspondientes salidas V-NN de primer $s^{(1)}$, segundo $s^{(2)}$ y tercer $s^{(3)}$ orden, para la base de datos Facial Recognition Technology (FERET). Los experimentos fueron realizados utilizando el modelo $40\mathrm{MLP}_{37,37,1}$, entrenado con la información de 40 sujetos de la base de datos FERET.

	$40\mathrm{MLP}_{37,37,1}$									
$\mathrm{RR}_i[\%]$	RR_1	RR_2	RR_3	RR_4	RR_5	RR_6	RR_7	RR_8	RR_9	RR_{10}
$a\mathrm{MLP}_{NI,NH,NO}$	97,16	97,50	100,00	96,25	97,73	97,61	97,50	97,50	98,52	97,95
$40\text{V-NN}_{s^{(1)}}$	100,00	95,11	97,50	95,00	97,05	95,00	99,09	94,20	95,45	95,00
$40\text{V-NN}_{s^{(2)}}$	97,50	93,64	98,41	97,73	95,45	96,14	97,05	94,55	98,18	95,00
$40\text{V-NN}_{s^{(3)}}$	93,07	93,86	100,00	96,59	97,73	96,48	96,02	96,36	97,16	95,00
$\mathrm{RR}_i[\%]$	RR_{11}	RR_{12}	RR_{13}	RR_{14}	RR_{15}	RR_{16}	RR_{17}	RR_{18}	RR_{19}	RR_{20}
$a\mathrm{MLP}_{NI,NH,NO}$	97,95	97,73	94,66	99,55	98,30	97,50	96,36	97,50	97,50	96,82
$40\text{V-NN}_{s^{(1)}}$	95,91	99,89	96,93	99,43	98,86	95,23	97,27	95,00	95,00	95,00
$40\text{V-NN}_{s^{(2)}}$	94,43	98,30	95,00	97,05	99,09	97,39	95,00	96,59	95,34	93,98
$40\text{V-NN}_{s^{(3)}}$	95,00	99,77	95,00	95,34	98,41	98,98	93,52	95,00	95,00	94,77
$\mathrm{RR}_i[\%]$	RR_{21}	RR_{22}	RR_{23}	RR_{24}	RR_{25}	RR_{26}	RR_{27}	RR_{28}	RR_{29}	RR_{30}
$a\mathrm{MLP}_{NI,NH,NO}$	97,50	97,50	97,50	97,50	97,50	97,50	97,50	97,50	95,23	97,50
$40\text{V-NN}_{s^{(1)}}$	98,98	94,66	97,61	95,91	99,43	96,70	95,00	98,52	99,55	95,00
$40\text{V-NN}_{s^{(2)}}$	98,41	95,00	97,05	96,93	99,32	94,43	95,00	95,45	94,89	95,00
$40\text{V-NN}_{s^{(3)}}$	94,20	97,05	98,75	96,48	97,39	94,43	95,00	96,14	98,41	95,00
$\mathrm{RR}_i[\%]$	RR_{31}	RR_{32}	RR_{33}	RR_{34}	RR_{35}	RR_{36}	RR_{37}	RR_{38}	RR_{39}	RR_{40}
$a\mathrm{MLP}_{NI,NH,NO}$	95,11	92,84	91,59	97,50	97,27	97,39	97,50	96,82	98,52	97,50
$40\text{V-NN}_{s^{(1)}}$	97,95	97,73	94,32	96,36	95,00	95,23	99,43	98,30	95,00	95,11
$40\text{V-NN}_{s^{(2)}}$	94,66	97,73	96,25	97,84	96,59	95,57	99,66	98,30	94,09	95,80
$40\text{V-NN}_{s^{(3)}}$	97,95	98,07	95,68	96,36	96,36	97,50	93,64	94,32	95,45	96,25

Tabla suplementaria 4 Comparación de las tasas de reconocimiento global (RR) y de ahorro de espacio (SS) para un clasificador aV-NN (a=40) y sus correspondientes salidas V-NN de primer $s^{(1)}$, segundo $s^{(2)}$ y tercer $s^{(3)}$ orden, para la base de datos Facial Recognition Technology (FERET). Los experimentos fueron realizados utilizando un modelo $40\mathrm{MLP}_{37,37,1}$, representando 40 sujetos de la base de datos FERET.

		$40MLP_{37,37,1}$	
$P_{aMLP_{NI,NH,NO}} \rightarrow$		59200	
$\mathrm{RR}[\%] \rightarrow$		97,16	
	$P_{as^{(i)}}$	RR[%]	SS[%]
$40\text{V-NN}_{s^{(1)}}$	1480	96,69	97,50
$40\text{V-NN}_{s^{(2)}}$	29600	96,34	50,00
$40\text{V-NN}_{s^{(1)}}$	84360	96,37	−42,50

APÉNDICE B

Código de los experimentos realizados en la tesis

B.1. Modelo *a*V-NN para la compresión de clasificadores neuronales

Listado B.1: Módulo Volterra-NN

```
function [] = volterraNN(dataFoldName,hidNumber,CfrLevel,FoldNumber,RepeatNumber,
    classesNumber)
%
% 16/07/2010
% Mariano Rubiolo
%
%-------------------------------------------------------------------------
% INPUT
% dataFoldName: name of the data fold
% classesNumber: number of classes in the dataset
% RepeatNumber: number of repetition of each run
% FoldNumber: number of data folds
% CfrLevel: aceptance of recognition rate [%] for NN
% hidNumber: number of changes in the number of hidden neurons
% -------------------------------------------------------------------------

% TRAINING PARAMETERS
Epochs=100;
Goal=1e-10;
Show=50;

% Activation functions
IHActFunc = 'logsig'; %at hidden layer
HOActFunc = 'purelin'; % at output layer

% Training algorithm
trainFunc = 'trainlm'; % Levenberg Marquardt algorithm

```

```
% Hidden layer parameters
startHidNumber = 1;
hidRate = 1;
%-------------------------------------------------------------------------
% For each fold...
for fold=1:FoldNumber

    % data file
    foldfile = [dataFoldName,int2str(fold),'.mat'];
    load(foldfile);

    % Train/Test data
    InputTrain=cell2mat(inputTrainCells(fold));
    InputTest=cell2mat(inputTestCells(fold));

    %Input/Output Normalization: 0-1
    [InData, xs] = mapminmax(InputTrain,0,1);
    [InDataTest, xsTest] = mapminmax(InputTest,0,1);

    % 'train set' classes limits
    lt = [];
    lt(1) = size(InData,2)/classesNumber;
    for cn = 2:classesNumber
         eval(['lt(',int2str(cn),') = lt(',int2str(cn-1),')+size(InData,2)/
             classesNumber;']);
    end;

    % 'test set' classes limits
    ls = [];
    ls(1) = size(InDataTest,2)/classesNumber;
    for cn = 2:classesNumber
         eval(['ls(',int2str(cn),') = ls(',int2str(cn-1),')+size(InDataTest,2)/
             classesNumber;']);
    end;

    % Regroup data by applying the "1st Known" criteria
    InData1 = InData;
    if (classesNumber > 2)
        for cn = 2:classesNumber-1
            eval(['InData',int2str(cn),' = [InData(:,lt(',int2str(cn-1),')+1:lt(',
                int2str(cn),')) InData(:,1:lt(',int2str(cn-1),')) InData(:,lt(',
                int2str(cn),')+1:lt(classesNumber))];']);
        end;
    end;
    eval(['InData',int2str(classesNumber),' = [InData(:,lt(',int2str(classesNumber
        -1),')+1:lt(',int2str(classesNumber),')) InData(:,1:lt(',int2str(
        classesNumber-1),')) ];']);

    InDataTest1 = InDataTest;
    if (classesNumber > 2)
        for cn = 2:classesNumber-1
```

```
73              eval(['InDataTest',int2str(cn),' = [InDataTest(:,ls(',int2str(cn-1),')
                    +1:ls(',int2str(cn),')) InDataTest(:,1:ls(',int2str(cn-1),'))
                    InDataTest(:,ls(',int2str(cn),')+1:ls(classesNumber))];']);
74          end;
75      end;
76      eval(['InDataTest',int2str(classesNumber),' = [InDataTest(:,ls(',int2str(
            classesNumber-1),')+1:ls(',int2str(classesNumber),')) InDataTest(:,1:ls(',
            int2str(classesNumber-1),')) ];']);
77
78
79      % Make target data (1 belonging to the class, 0 if not) by applying the "1st
            Known" criteria
80      OutData1=[ones(1,lt(1)) zeros(1,lt(classesNumber-1))];
81      for cn = 2:classesNumber
82          eval(['OutData',int2str(cn),'=OutData1;']);
83      end;
84
85      OutDataTest1=[ones(1,ls(1)) zeros(1,ls(classesNumber-1))];
86      for cn = 2:classesNumber
87          eval(['OutDataTest',int2str(cn),'=OutDataTest1;']);
88      end;
89
90      OutDataTest1NEW=[ones(1,ls(1)) zeros(1,ls(classesNumber-1))];
91      for cn = 2:classesNumber
92          eval(['OutDataTest',int2str(cn),'NEW = OutDataTest1NEW;']);
93      end;
94
95           % for each class
96           for i=1:classesNumber
97
98               % for each variant of hidden neuron number
99               for hidNeu=startHidNumber:hidNumber
100
101              stringHidNeu = ['HiddenNeurons = round (hidNeu*size(InData',int2str(i
                     ),',1)*hidRate);'];
102              eval(stringHidNeu);
103
104              AcceptedAux = 1;  % acceptance condition
105
106              %-------------------------------------------------------------
107              % Class 'i'
108              while (AcceptedAux < RepeatNumber+1)
109
110                     % Creating the NN
111                     stringConfMLP = ['MLP',int2str(i),'= newff(minmax(InData',
                            int2str(i),'),[HiddenNeurons size(OutData',int2str(i),',1)
                            ],{ IHActFunc HOActFunc},trainFunc);'];
112                     eval(stringConfMLP);
113
114                     stringParameters = ['MLP',int2str(i),'.trainParam.epochs =
                            Epochs;MLP',int2str(i),'.trainParam.goal = Goal;MLP',
```

```
                        int2str(i),'.trainParam.show = Show;'];
115                   eval(stringParameters);
116
117                   %Training the NN
118                   stringTrain = ['[MLP',int2str(i),',trMLP',int2str(i),'] =
                          train(MLP',int2str(i),',InData',int2str(i),',OutData',
                          int2str(i),');'];
119                   eval(stringTrain);
120
121
122                    eval(['NNout_test',int2str(i),' = sim(MLP',int2str(i),',
                           InDataTest',int2str(i),');']);
123
124                    eval(['CfR_MLP',int2str(i),' = (size(find((round(NNout_test',
                           int2str(i),')-OutDataTest',int2str(i),'NEW)==0),2))/size(
                           OutDataTest',int2str(i),'NEW,2)*100']);
125
126                    eval(['CfR_MLP_aux = CfR_MLP',int2str(i)]);
127
128
129                    % If the MLP has a classification/recognition rate >=
                           CfrLevel...
130                    if (CfR_MLP_aux >= CfrLevel)
131
132                       %-----------------------------------------------
133                       % Volterra model building - for training data
134                       eval(['[Volterra1_C',int2str(i),', Volterra2_C',int2str(i)
                              ,', Volterra3_C',int2str(i),', VolterraOut_C',int2str(
                              i),'] = volterraModelBuilding(MLP',int2str(i),',InData
                              ',int2str(i),');']);
135                       %-----------------------------------------------
136                       % Volterra model building - for test data
137                       eval(['[Volterra1_Sim_C',int2str(i),', Volterra2_Sim_C',
                              int2str(i),', Volterra3_Sim_C',int2str(i),',
                              VolterraOut_Sim_C',int2str(i),'] =
                              volterraModelBuilding(MLP',int2str(i),',InDataTest',
                              int2str(i),');']);
138                       %-----------------------------------------------
139
140                       %-----------------------------------------------
141                       %Volterra 1st order approximation
142                       eval(['[threshold1_V1C',int2str(i),',threshold2_V1C',
                              int2str(i),',rtaV1_C',int2str(i),',Volterra1_C',
                              int2str(i),'Norm,V1_C',int2str(i),',CfR_V1_C',int2str(
                              i),'] = volterraSimulation(lt,ls,Volterra1_C',int2str(
                              i),',Volterra1_Sim_C',int2str(i),',OutDataTest',
                              int2str(i),'NEW);']);
143                       %-----------------------------------------------
144                       %Volterra 2nd order approximation
145                       eval(['[threshold1_V2C',int2str(i),',threshold2_V2C',
                              int2str(i),',rtaV2_C',int2str(i),',Volterra2_C',
```

```
                                int2str(i),'1Norm,V2_C',int2str(i),',CfR_V2_C',int2str
                                (i),'] = volterraSimulation(lt,ls,Volterra2_C',int2str
                                (i),',Volterra2_Sim_C',int2str(i),',OutDataTest',
                                int2str(i),'NEW);']);
146                         %-----------------------------------------------
147                         %Volterra 3rd order approximation
148                         eval(['[threshold1_V3C',int2str(i),',threshold2_V3C',
                                int2str(i),',rtaV3_C',int2str(i),',Volterra3_C',
                                int2str(i),'Norm,V3_C',int2str(i),',CfR_V3_C',int2str(
                                i),'] = volterraSimulation(lt,ls,Volterra3_C',int2str(
                                i),',Volterra3_Sim_C',int2str(i),',OutDataTest',
                                int2str(i),'NEW);']);
149                         %-----------------------------------------------
150
151                         % Classification/Recognition rates calculus for MLP output
                                , firs, second and third order Volterra output.
152                         eval(['CfR_MLP_C',int2str(i),'_F',int2str(fold),'_H',
                                int2str(HiddenNeurons),'_rep',int2str(AcceptedAux),' =
                                 CfR_MLP',int2str(i)]);
153                         eval(['CfR_V1_C',int2str(i),'_F',int2str(fold),'_H',
                                int2str(HiddenNeurons),'_rep',int2str(AcceptedAux),' =
                                 CfR_V1_C',int2str(i)]);
154                         eval(['CfR_V2_C',int2str(i),'_F',int2str(fold),'_H',
                                int2str(HiddenNeurons),'_rep',int2str(AcceptedAux),' =
                                 CfR_V2_C',int2str(i)]);
155                         eval(['CfR_V3_C',int2str(i),'_F',int2str(fold),'_H',
                                int2str(HiddenNeurons),'_rep',int2str(AcceptedAux),' =
                                 CfR_V3_C',int2str(i)]);
156
157                         % Accepting experiment...
158                         AcceptedAux = AcceptedAux+1;
159
160                         % Saving experiment...
161                         savefile = ['Resultados_letterUCI_',int2str(classesNumber)
                                ,'VNNs_rep',int2str(RepeatNumber),'_CS.mat'];
162                         save(savefile);
163                     end;
164             end;
165         end;
166     end;
167 end;
```

Listado B.2: Módulo Volterra model building

```
1 function [VOLT1,VOLT2,VOLT3,VOLTout] = volterraModelBuilding(MLP,InData)
2 % ------------------------------------------------------------------------
3 % 22/06/2010
4 % Mariano Rubiolo / Georgina Stegmayer
5 % ------------------------------------------------------------------------
6 % INPUT PARAMETERS
7 % HO,H1,H2,H3: n-order volterra kernels
```

```
% Indata: input data from what the volterra model is builded
% I_N: number of input neurons
%
% ------------------------------------------------------------------------
% OUTPUT PARAMETERS
% VOLT1,VOLT2,VOLT3: n-order volterra approximation
% VOLTout: model output (highest order volterra approximation)
%------------------------------------------------------------------------

%------------------------------------------------------------------------
I_N = size(MLP.IW{1,1},2);  % number of input neurons
H_N = size(MLP.IW{1,1},1);  % number of hidden neurons
%------------------------------------------------------------------------

%------------------------------------------------------------------------
% ANN parameters extraction
WIJ = MLP.IW{1,1};    % input weights matrix
W = MLP.LW{2,1};      % hidden neurons weights matrix
B = MLP.b{1};         % hidden neurons bias matrix


%------------------------------------------------------------------------
% VOLTERRA KERNELES CALCULUS from ANN parameters
%------------------------------------------------------------------------
% H0
H0 = MLP.b{2}(1);     % initialize H0= b0
for j = 1:H_N
    H0 = H0 + W(j).* [( 1/(1 + exp(-(B(j)))))];  % impulse response, kernel HO
end;
%------------------------------------------------------------------------
% matrix that contains the 1st. order kernels H1
H1=zeros(1,I_N);
for i = 1:I_N,
   for j = 1:H_N,
      H1(i) = H1(i) + W(j) * WIJ(j,i) * [(exp(-B(j))/(1+exp(-(B(j))))^2 )];
   end;
end;
% H1 = [ h1(0) h1(1) h1(2) ... ]
%------------------------------------------------------------------------
% matrix that contains the 2nd. order kernels H2
H2=zeros(I_N,I_N);
for i = 1:I_N,
    for k = 1:I_N,
       for j = 1:H_N,
          H2(i,k) = H2(i,k) + W(j) * WIJ(j,i) * WIJ(j,k) * [( exp(-B(j))*(exp(-B(j
              ))-1)/(1+exp(-B(j)))^3 ) / factorial(2)];
       end;
    end;
end;
% H2 = [ h2(0,0) h2(0,1) h2(0,2) ...
%        h2(1,0) h2(1,1) h2(1,2) ...
```

```
%        h2(2,0) h2(2,1) h2(2,2) ... ]
%-------------------------------------------------------------------------

% matrix that contains the 3rd. order kernels h3
H3=zeros(I_N,I_N);
for i = 1:I_N,
    for k = 1:I_N,
        for j = 1:H_N
            if i == k
               H3(i,k) = H3(i,k) + W(j) * WIJ(j,i) * WIJ(j,i) * WIJ(j,k) * [( -exp
                    (-B(j))*(-exp(-2*B(j))+4*exp(-B(j))-1)/((1+exp(-B(j)))^4) ) /
                    factorial(3)];
             else
               H3(i,k) = H3(i,k) +  3* W(j) * WIJ(j,i) * WIJ(j,i) * WIJ(j,k) * [(
                    -exp(-B(j))*(-exp(-2*B(j))+4*exp(-B(j))-1)/((1+exp(-B(j)))^4) )
                     / factorial(3)];
             end;
        end;
    end;
end;
% H3 = [ h3(0,0,0) h3(0,0,1) h3(0,0,2) ...
%        h3(1,1,0) h3(1,1,1) h3(1,1,2) ...
%        h3(2,2,0) h3(2,2,1) h3(2,2,2) ...]

%-------------------------------------------------------------------------
% VOLTERRA MODEL BUILDING PROCESS
%-------------------------------------------------------------------------
VOLT1=H0;
for i = 1:I_N,
    VOLT1= VOLT1 + InData(i,:).*H1(i);  % 1st. order approximation
end;

VOLT2=VOLT1;
for i = 1:I_N,
    for j = 1:I_N,
        VOLT2= VOLT2 + InData(i,:) .* InData(j,:) .* H2(i,j);  % 2nd. order
            approximation
    end;
end;

VOLT3=VOLT2;
for i = 1:I_N,
    for j = 1:I_N,
        VOLT3= VOLT3 + InData(i,:) .* InData(i,:) .* InData(j,:) .* H3(i,j);  % 3rd
            . order approximation
    end;
end;

VOLTout = VOLT3;
```

Listado B.3: Módulo Volterra simulation

```
function [threshold1,threshold2,answer,VOLTtrainNorm,VOLTsimNorm,CfR] =
    volterraSimulation(lt,ls,VOLTtrain,VOLTsim,OutputTest)
%-------------------------------------------------------------------------
% 22/06/2010
% Mariano Rubiolo / Georgina Stegmayer
%
%-------------------------------------------------------------------------
% INPUT PARAMETERS
% lt: train set limits
% ls: simulate set limits
% VOLTtrain: n-order volterra approximation for training data
% VOLTsim: n-order volterra approximation for simulation/testing data
% OutputTest: output data set
%
%-------------------------------------------------------------------------
% OUTPUT PARAMETERS
% threshold1: threshold between class 1 and 2
% threshold2: threshold between class 2 and 3
% answer: classification output
% VOLTsim: model output (highest order volterra approximation)
% CfR: Classification/recognition rate
%-------------------------------------------------------------------------

%NORMALIZATION
[VOLTtrainNorm, xt] = mapminmax(VOLTtrain,0,1);
[VOLTsimNorm, xs] = mapminmax(VOLTsim,0,1);

%-------------------------------------------------------------------------
% Thresholds calculation for Volterra n-order approximation (train set)
% 1st class - n-order approximation
minmaxC1 = minmax(VOLTtrainNorm(1:lt(1)));
minC1 = minmaxC1(1); %min clase 1
maxC1 = minmaxC1(2); %min clase 1

%Threshold for classification
threshold1=minC1;
threshold2=maxC1;

%-------------------------------------------------------------------------
% Volterra n-order approximation (test set)
answer=0;
for j=1:length(VOLTsimNorm)
    if ((VOLTsimNorm(j)>= threshold1) && (VOLTsimNorm(j)<= threshold2))
        answer(j)=1; %  class 1
    else
        answer(j)=0;    % error
   end
end

% number of erroneous clasification
```

```
50 bad = [find(answer(1:ls(1))~=OutputTest(ls(1))) find(answer(ls(1)+1:ls(2))~=
       OutputTest(ls(1)+1)) find(answer(ls(2)+1:ls(3))~= OutputTest(ls(2)+1))];
51
52 % Classification/Recognition error calculus
53 CfR = 100-length(bad)*100/length(OutputTest);
```

B.2. Clasificador neuronal para correspondencias entre ontologías

Listado B.4: Módulo Entrenamiento aleatorio. Ejemplo para K1.

```
%--------------------------------------------------------------------------
% 1 RED - ENTRENAMIENTO - ALEATORIO - K1
%--------------------------------------------------------------------------
clear all
clc

ResultadoTotalPromedio = cell(41,4);
ResultadoTotalPromedio(1,1)={'KA'};
ResultadoTotalPromedio(1,2)={'SWRC'};
ResultadoTotalPromedio(1,3)={'OLID'};
ResultadoTotalPromedio(1,4)={'SINC'};

for i=1:40
    ResultadoTotalPromedio(i+1,1)={0};
    ResultadoTotalPromedio(i+1,2)={0};
    ResultadoTotalPromedio(i+1,3)={0};
    ResultadoTotalPromedio(i+1,4)={0};
end

PerformancePromedio=0;
AcumuladoMinimoKA=0;
AcumuladoMinimoSWRC=0;
AcumuladoMinimoOLID=0;
AcumuladoMinimoSINC=0;

mkdir K1

cd ..
cd ..
cd Patrones

Parametros
PatronFormacionK1
TargetKA
TargetSWRC
TargetOLID
TargetSINC

cd ..
cd 1RN
cd Aleatorio

[pn,ps] = mapminmax(EntrenamientoK1,0,1);
```

```
Esperado = [EsperadoKA; EsperadoSWRC; EsperadoOLID;EsperadoSINC];
EntradaKA = pn(:,(1:43));
EntradaSWRC = pn(:,(44:86));
EntradaOLID = pn(:,(87:129));
EntradaSINC = pn(:,(130:172));
n=4;

indicesAleatorios = zeros(1,172);
a = fix(rand*172)+1;
indicesAleatorios(:,1) = a;

indice = indicesAleatorios(1);

Entrada(:,1) = pn(:,indice);
Salida(:,1) = Esperado(:,indice);
for columna=2:172
    a = fix(rand*172)+1;
    while any(indicesAleatorios == a) == 1
        a = fix(rand*172)+1;
    end
    indicesAleatorios(:,columna) = a;

    indice = indicesAleatorios(columna);

    Entrada(:,columna) = pn(:,indice);
    Salida(:,columna) = Esperado(:,indice);

end

j=1;
while j<=Iteraciones,

NN=newff(minmax(Entrada),[H n],{'logsig' 'logsig'});
NN.trainParam.epochs = Epocas;
[NN,tr,SalidaObtenida,Error]=train(NN,Entrada,Salida);
PerformancePromedio= PerformancePromedio + mse(Error);
close all;

%--------------------------------------------------------------------------
% PRUEBAS
%--------------------------------------------------------------------------

Resultado = cell(41,4);
Resultado(1,1)={'KA'};
Resultado(1,2)={'SWRC'};
Resultado(1,3)={'OLID'};
Resultado(1,4)={'SINC'};


for i=1:40
```

```
95
96      Patron = PruebaK1(:,i);
97      Patron = mapminmax('apply',Patron,ps);
98
99      ResultadoParcial = sim(NN,Patron)';
100     Resultado(i+1,1)={ResultadoParcial(1)};
101     Resultado(i+1,2)={ResultadoParcial(2)};
102     Resultado(i+1,3)={ResultadoParcial(3)};
103     Resultado(i+1,4)={ResultadoParcial(4)};
104
105
106     Agregado1 = ResultadoTotalPromedio{i+1,1} + Resultado{i+1,1};
107     Agregado2 = ResultadoTotalPromedio{i+1,2} + Resultado{i+1,2};
108     Agregado3 = ResultadoTotalPromedio{i+1,3} + Resultado{i+1,3};
109     Agregado4 = ResultadoTotalPromedio{i+1,4} + Resultado{i+1,4};
110
111     ResultadoTotalPromedio(i+1,1)={Agregado1};
112     ResultadoTotalPromedio(i+1,2)={Agregado2};
113     ResultadoTotalPromedio(i+1,3)={Agregado3};
114     ResultadoTotalPromedio(i+1,4)={Agregado4};
115
116 end
117
118 Archivo = ['K1\Aleatorio_K1_It',num2str(j)];
119 save(Archivo);
120
121 ResultadoKA = sim(NN,EntradaKA);
122 MinimoIteracionKA = min(ResultadoKA(1,:));
123 AcumuladoMinimoKA = AcumuladoMinimoKA + MinimoIteracionKA;
124
125 ResultadoSWRC = sim(NN,EntradaSWRC);
126 MinimoIteracionSWRC = min(ResultadoSWRC(2,:));
127 AcumuladoMinimoSWRC = AcumuladoMinimoSWRC + MinimoIteracionSWRC;
128
129 ResultadoOLID = sim(NN,EntradaOLID);
130 MinimoIteracionOLID = min(ResultadoOLID(3,:));
131 AcumuladoMinimoOLID = AcumuladoMinimoOLID + MinimoIteracionOLID;
132
133 ResultadoSINC = sim(NN,EntradaSINC);
134 MinimoIteracionSINC = min(ResultadoSINC(4,:));
135 AcumuladoMinimoSINC = AcumuladoMinimoSINC + MinimoIteracionSINC;
136
137 j=j+1;
138 end;
139
140 for i=1:40
141     ResultadoTotalPromedio(i+1,1)={ResultadoTotalPromedio{i+1,1}/Iteraciones};
142     ResultadoTotalPromedio(i+1,2)={ResultadoTotalPromedio{i+1,2}/Iteraciones};
143     ResultadoTotalPromedio(i+1,3)={ResultadoTotalPromedio{i+1,3}/Iteraciones};
144     ResultadoTotalPromedio(i+1,4)={ResultadoTotalPromedio{i+1,4}/Iteraciones};
145 end
```

```
146
147 PerformancePromedio =PerformancePromedio/Iteraciones;
148 MinimoKA = AcumuladoMinimoKA/Iteraciones;
149 MinimoSWRC = AcumuladoMinimoSWRC/Iteraciones;
150 MinimoOLID = AcumuladoMinimoOLID/Iteraciones;
151 MinimoSINC = AcumuladoMinimoSINC/Iteraciones;
152
153
154 archivo = 'K1\Aleatorio_K1_Promedio';
155 save(archivo);
156 xlswrite('OcurrenciasAleatorio.xls', ResultadoTotalPromedio, 'K1', 'a106:d146');
157 xlswrite('OcurrenciasAleatorio.xls', PerformancePromedio, 'K1', 'g104');
158 xlswrite('OcurrenciasAleatorio.xls', MinimoKA, 'K1', 'g108');
159 xlswrite('OcurrenciasAleatorio.xls', MinimoSWRC, 'K1', 'h108');
160 xlswrite('OcurrenciasAleatorio.xls', MinimoOLID, 'K1', 'i108');
161 xlswrite('OcurrenciasAleatorio.xls', MinimoSINC, 'K1', 'j108');
```

Listado B.5: Módulo Entrenamiento 1 kn, 3 unkn. Ejemplo para K1.

```
 1 %--------------------------------------------------------------------------
 2 % 1 RED - ENTRENAMIENTO - UNO CONOCE TRES NO CONOCE - K1
 3 %--------------------------------------------------------------------------
 4 clear all
 5 clc
 6
 7 ResultadoTotalPromedio = cell(41,4);
 8 ResultadoTotalPromedio(1,1)={'KA'};
 9 ResultadoTotalPromedio(1,2)={'SWRC'};
10 ResultadoTotalPromedio(1,3)={'OLID'};
11 ResultadoTotalPromedio(1,4)={'SINC'};
12
13 for i=1:40
14     ResultadoTotalPromedio(i+1,1)={0};
15     ResultadoTotalPromedio(i+1,2)={0};
16     ResultadoTotalPromedio(i+1,3)={0};
17     ResultadoTotalPromedio(i+1,4)={0};
18 end
19
20 PerformancePromedio=0;
21 AcumuladoMinimoKA=0;
22 AcumuladoMinimoSWRC=0;
23 AcumuladoMinimoOLID=0;
24 AcumuladoMinimoSINC=0;
25
26 mkdir K1
27
28 cd ..
29 cd ..
30 cd Patrones
31
32 Parametros
```

```
PatronFormacionK1
TargetKA
TargetSWRC
TargetOLID
TargetSINC

cd ..
cd 1RN
cd UnoConoceTresNoConoce

[pn,ps] = mapminmax(EntrenamientoK1,0,1);
Esperado = [EsperadoKA; EsperadoSWRC; EsperadoOLID; EsperadoSINC];
EntradaKA = pn(:,(1:43));
EntradaSWRC = pn(:,(44:86));
EntradaOLID = pn(:,(87:129));
EntradaSINC = pn(:,(130:172));
n=4;

columna=1;
for i=1:43

    Entrada(:,columna) = pn(:,i);
    Salida(:,columna) = Esperado(:,i);
    columna=columna+1;

    Entrada(:,columna) = pn(:,i+43);
    Salida(:,columna) = Esperado(:,i+43);
    columna=columna+1;

    Entrada(:,columna) = pn(:,i+86);
    Salida(:,columna) = Esperado(:,i+86);
    columna=columna+1;

    Entrada(:,columna) = pn(:,i+129);
    Salida(:,columna) = Esperado(:,i+129);
    columna=columna+1;
end

j=1;
while j<=Iteraciones,

NN=newff(minmax(Entrada),[H n],{'logsig' 'logsig'});
NN.trainParam.epochs = Epocas;
[NN,tr,SalidaObtenida,Error]=train(NN,Entrada,Salida);
PerformancePromedio= PerformancePromedio + mse(Error);
close all;

%--------------------------------------------------------------------------
% PRUEBAS
%--------------------------------------------------------------------------

```

```
Resultado = cell(341,4);
Resultado(1,1)={'KA'};
Resultado(1,2)={'SWRC'};
Resultado(1,3)={'OLID'};
Resultado(1,4)={'SINC'};


for i=1:40

    Patron = PruebaK1(:,i);
    Patron = mapminmax('apply',Patron,ps);

    ResultadoParcial = sim(NN,Patron)';
    Resultado(i+1,1)={ResultadoParcial(1)};
    Resultado(i+1,2)={ResultadoParcial(2)};
    Resultado(i+1,3)={ResultadoParcial(3)};
    Resultado(i+1,4)={ResultadoParcial(4)};


    Agregado1 = ResultadoTotalPromedio{i+1,1} + Resultado{i+1,1};
    Agregado2 = ResultadoTotalPromedio{i+1,2} + Resultado{i+1,2};
    Agregado3 = ResultadoTotalPromedio{i+1,3} + Resultado{i+1,3};
    Agregado4 = ResultadoTotalPromedio{i+1,4} + Resultado{i+1,4};

    ResultadoTotalPromedio(i+1,1)={Agregado1};
    ResultadoTotalPromedio(i+1,2)={Agregado2};
    ResultadoTotalPromedio(i+1,3)={Agregado3};
    ResultadoTotalPromedio(i+1,4)={Agregado4};

end

Archivo = ['K1\UnoConoceTresNoConoce_K1_It',num2str(j)];
save(Archivo);

ResultadoKA = sim(NN,EntradaKA);
MinimoIteracionKA = min(ResultadoKA(1,:));
AcumuladoMinimoKA = AcumuladoMinimoKA + MinimoIteracionKA;

ResultadoSWRC = sim(NN,EntradaSWRC);
MinimoIteracionSWRC = min(ResultadoSWRC(2,:));
AcumuladoMinimoSWRC = AcumuladoMinimoSWRC + MinimoIteracionSWRC;

ResultadoOLID = sim(NN,EntradaOLID);
MinimoIteracionOLID = min(ResultadoOLID(3,:));
AcumuladoMinimoOLID = AcumuladoMinimoOLID + MinimoIteracionOLID;

ResultadoSINC = sim(NN,EntradaSINC);
MinimoIteracionSINC = min(ResultadoSINC(4,:));
AcumuladoMinimoSINC = AcumuladoMinimoSINC + MinimoIteracionSINC;

j=j+1;
```

```
135 end;
136
137 for i=1:40
138     ResultadoTotalPromedio(i+1,1)={ResultadoTotalPromedio{i+1,1}/Iteraciones};
139     ResultadoTotalPromedio(i+1,2)={ResultadoTotalPromedio{i+1,2}/Iteraciones};
140     ResultadoTotalPromedio(i+1,3)={ResultadoTotalPromedio{i+1,3}/Iteraciones};
141     ResultadoTotalPromedio(i+1,4)={ResultadoTotalPromedio{i+1,4}/Iteraciones};
142 end
143
144 PerformancePromedio =PerformancePromedio/Iteraciones;
145 MinimoKA = AcumuladoMinimoKA/Iteraciones;
146 MinimoSWRC = AcumuladoMinimoSWRC/Iteraciones;
147 MinimoOLID = AcumuladoMinimoOLID/Iteraciones;
148 MinimoSINC = AcumuladoMinimoSINC/Iteraciones;
149
150 archivo = 'K1\UnoConoceTresNoConoce_K1_Promedio';
151 save(archivo);
152 xlswrite('OcurrenciasUnoConoceTresNoConoce.xls', ResultadoTotalPromedio, 'K1', '
       a106:d146');
153 xlswrite('OcurrenciasUnoConoceTresNoConoce.xls', PerformancePromedio, 'K1', 'g104
       ');
154 xlswrite('OcurrenciasUnoConoceTresNoConoce.xls', MinimoKA, 'K1', 'g108');
155 xlswrite('OcurrenciasUnoConoceTresNoConoce.xls', MinimoSWRC, 'K1', 'h108');
156 xlswrite('OcurrenciasUnoConoceTresNoConoce.xls', MinimoOLID, 'K1', 'i108');
157 xlswrite('OcurrenciasUnoConoceTresNoConoce.xls', MinimoSINC, 'K1', 'j108');
```

B.3. Redes neuronales para el modelado de series temporales

Listado B.6: Módulo Modeling gen-to-gen relation by NN

```
% Evaluate the gene i as the potential regulator of gene j.
% All possible genes at 'ies' set are involved.
%
% Mariano Rubiolo
%
% INPUTS
% iInit: initial gene in the set
% iFinal: final gene in the set
% ies: set of genes as possible regulators
% j: regulated gene
% dT_i: time delay considered at the input gene
% dT_j: time delay considered at the output target (default: 0)
% dT_o: time delay considered at the other genes in the set(default: 0)
% dataSet: data
% repNumber: number of repetitions of the experiment
% realNumber: number of realizations for training
% hidVar: variaton of the number of hidden neurons
% path: path of the output file
% Epochs: number of epochs for training

function [results] = NNTemporal_Ninit_wiGies_vs_wjGj_allArq(iInit,iFinal,ies,j,
      dT_i,dT_j,dT_o,dataSet,repNumber,realNumber,hidVar,path,Epochs)


% TRaining parameters configuration
Goal=1e-150;
MinGrad = 1e-100;
Show=50;
RepeatNumber = repNumber(2);
CfrLevel = 0;
results = zeros(2,size(ies,2));

% For all genes in the candidates set...
 for i=iInit:iFinal

     % delta T input gene
     for w=1:size(dT_i,2)
         w_i = dT_i(w);

         % delta T target gene
         for w2 = 1:size(dT_j,2);
             w_j = dT_j(w2);

```

```
43              % delta T ohter genes
44              for w3 = size(dT_o,2);
45                  w_o = dT_o(w3);
46
47                  % Hidden number variation
48                  for hidNeu=hidVar(1):hidVar(2)
49
50                      % particular data set creation
51                      dataSetFile = [dataSet,'_1.mat'];
52                      [inputSet1, targetSet1] = funcCreateTSdata_allArq(w_i,w_j,w_o
                            ,ies(i),j,setdiff(ies,[ies(i),j]),dataSetFile);
53
54                      % INPUTs and OUTPUTs for the model
55                      [inputData1, xsI1] = mapminmax(inputSet1,0,1);
56                      [targetData1, xsT1] = mapminmax(targetSet1,0,1);
57
58                      InData1 = inputData1;
59                      InDataTest1 = inputData1;
60                      OutData1 = targetData1;
61                      OutDataTest1 = OutData1;
62                      OutDataTest1NEW=OutDataTest1;
63
64                      HiddenNeurons = round (hidNeu*size(InData1,1)*0.5);
65
66                      %  Class i %
67
68                      Accepted = repNumber(1);
69
70                      while ((Accepted) < repNumber(2)+1)
71
72                          savefile = [path,'Rs_NNTemp_w',int2str(w_i),'G',int2str(
                                ies(i)),'_vs_w',int2str(w_j),'G',int2str(j),'_incl',
                                int2str(w_o),'oG_',int2str(HiddenNeurons),'H_rep',
                                int2str(Accepted),'_',dataSet,'_',int2str(realNumber)
                                ,'real','.mat']
73
74                          Accepted1 = 0;
75
76                          while (Accepted1 == 0)
77
78                          % Creating the MLP
79                          MLP= newff(minmax(InData1),[HiddenNeurons size(OutData1
                                ,1)],{'logsig' 'purelin'},'trainlm');
80
81                          MLP.trainParam.epochs = Epochs;MLP.trainParam.goal = Goal
                                ;MLP.trainParam.show = Show; MLP.trainParam.min_grad
                                = MinGrad;  %MLP.trainParam.mu_inc = Mu_inc;
82
83                          % Training the MLP
84                          [MLP,trMLP] = train(MLP,InData1,OutData1);
85
```

```
86                          % realization numbers
87                          for rz = 2:realNumber
88
89                            % data set creation for the next realizations
90                            dataSetFile = [dataSet,'_',int2str(rz),'.mat'];
91                            string1 = ['[inputSet',int2str(rz),',targetSet',int2str
                                  (rz),'] = funcCreateTSdata_allArq(w_i,w_j,w_o,ies(i
                                  ),j,setdiff(ies,[ies(i),j]),dataSetFile);'];
92                            eval(string1);
93                            string2 = ['[inputData',int2str(rz),', xsI',int2str(rz)
                                  ,'] = mapminmax(inputSet',int2str(rz),',0,1);'];
94                            eval(string2);
95                            string3 = ['[targetData',int2str(rz),', xsT',int2str(rz
                                  ),'] = mapminmax(targetSet',int2str(rz),',0,1);'];
96                            eval(string3);
97                            string4 = ['[MLP,trMLP] = train(MLP,inputData',int2str(
                                  rz),',targetData',int2str(rz),');'];
98                            eval(string4);
99                          end;
100
101
102                         % Testing the MLP
103                         NNout_test = sim(MLP,InDataTest1);
104                         MSE = trMLP.perf(size(trMLP.perf,2));
105                         CfR_MLP = (size(find((round(NNout_test)- round(
                                OutDataTest1NEW))==0),2))/size(OutDataTest1NEW,2)*100;
106
107                         % Results matrix
108                         if CfR_MLP>= CfrLevel,
109                               results(i,1) = ies(i);
110                               results(i,2) = j;
111                               results(i,3) = w;
112                               results(i,4) = w_j;
113                               results(i,5) = HiddenNeurons;
114                               results(i,6) = MSE;
115                               Accepted1 = 1;
116                           end;
117
118                           end;
119                           Accepted = Accepted+1;
120                           save(savefile)
121
122
123                   end;
124               end;
125             end;
126         end;
127     end;
128 end;
```

Listado B.7: Módulo Create dataset for a relation gen-to-gen

```
% Creating the dataset for training NN with the information of gene i and gene j.
%
% Mariano Rubiolo
%
% INPUTS
% gen_i: gene input
% gen_j: gene target
% others: set of other genes as possible regulators
% w: time delay considered at the input gene
% w2: time delay considered at the output target (default: 0)
% w3: time delay considered at the other genes in the set(default: 0)
% dataSet: data
%

function [inputSet, targetSet, savefile] = funcCreateTSdata_allArq(w,w2,w3,gen_i,
      gen_j,others,dataset)

load(dataset);
data = data';
dSL = size(data,2);  % data Set Length

windowLength = w-1;
inputSet = zeros(dSL-windowLength,windowLength+1);
targetSet = zeros(dSL-windowLength,1);

for ti=0:(dSL-windowLength-1)
    for wl=1:windowLength+1  % until time t
        inputSet(ti+1,wl) = data(gen_i+1,wl+ti);
    end
    % the other genes at time t
    for wl3=1:w3+1  % si se requiere el tiempo t de los otros genes
        for pos=1:size(others,2)
            inputSet(ti+1,windowLength+pos+1) = data(others(pos)+1,windowLength+ti
                +1);
        end
    end
    % target gene
    for wl2=1:w2  % if the time t-1 of the target gene is required
        inputSet(ti+1,windowLength+1+pos+wl2) = data(gen_j+1,wl2+ti);
    end
    targetSet(ti+1,1)= data(gen_j+1,ti+1+windowLength);
end
inputSet = inputSet';
targetSet = targetSet';
```

Listado B.8: Módulo Applying rules

```
%
% Applying mining rules for discovering GRNs
```

```
3 %
4 % Mariano Rubiolo
5 %
6 % INPUT parameters:
7 % resultsDS: data set of the results of the pool of NN
8 % allG: list of genes
9 % pCriteria: points criteria
10 % rCriteria: ranking criteria
11 % hCriteria: hidden neurons criteria
12 % tP: threshold for points
13 % tM: threshold for min error
14 % a: alfa value
15 % repetitions
16 %
17 function [thrUncSimMinPoiT,thrUncSimMinRanT] = applyingRules(resultsDS,allG,
       pCriteria,rCriteria,hCriteria,tP,tM,a,repetitions);
18
19 load(resultsDS);
20
21 resultsTable = table;
22
23 genesTable = zeros(length(allG),length(allG));
24
25
26 [minMSETable] = generateMinMSETable(genesTable,resultsTable,allG,hCriteria);
27
28 poiT = generatePointsTable(genesTable,resultsTable,allG,pCriteria,hCriteria,
       repetitions);
29 ranT = generateRankingTable(genesTable,resultsTable,allG,rCriteria,hCriteria,
       repetitions);
30
31 simMinPoiT = generateSimetricPyETable(poiT, minMSETable,a);
32 simMinRanT = generateSimetricPyETable(ranT, minMSETable,a);
33
34 uncSimMinPoiT = generateUnchainedPTable(simMinPoiT);
35 uncSimMinRanT = generateUnchainedPTable(simMinRanT);
36
37 thrUncSimMinPoiT = generateThresholdPyETable(uncSimMinPoiT,minMSETable,tP,tM);
38 thrUncSimMinRanT = generateThresholdPyETable(uncSimMinRanT,minMSETable,tP,tM);
```

Listado B.9: Módulo Generating MinMSE Table

```
1 %
2 % Generate a table with the minimun and the average MSE of the relation between
       genes after the NN training.
3 %
4 % Mariano Rubiolo
5 %
6 % Inputs parameters...
7 % genesTable: table of genes relations
8 % resultsTable: table of results. It must be orderder from the minimum to the
```

```
      maximun MSE.
% allG: list of genes
% hCriteria: hidden neurons criteria
%
% Output: minMSETable,meanMSETable
%

function[minMSETable,meanMSETable] = generateMinMSETable(genesTable,resultsTable,
      allG,hCriteria);

minMSETable = genesTable;
for GiIdx=1:size(allG,2)
    for GjIdx=1:size(allG,2)
        if GiIdx ~= GjIdx
           Gi = allG(GiIdx)
           Gj = allG(GjIdx)

         %obtiene los indices de las filas donde la columna J tiene el valor Gj
         colIesIdx = find(resultsTable(:,1)==Gi);
         colJesIdx = find(resultsTable(:,2)==Gj);

         parIdx = intersect(colIesIdx,colJesIdx);

         % obtiene los indices de las filas con los valores de H seleccionados
         evalStr=['colHesIdx = find(resultsTable(:,6)',hCriteria,');'];
         eval(evalStr);

         % obtiene los indices de las filas a evaluar
         filterIdx = intersect(parIdx,colHesIdx);

         subTable = resultsTable(filterIdx,:);
         size(subTable)

         minMSETable(GiIdx,GjIdx) = subTable(1,7);
         meanMSETable(GiIdx,GjIdx) = mean(subTable(:,7));

        end
    end
end
```

Listado B.10: Módulo Generating Points Table

```
% Generate the points criteria table.
%
% Mariano Rubiolo
%
% Input parameters...
% genesTable: table of genes relations.
% resultsTable: table of results.
% allG: list of genes.
% pCriteria: points criteria [1,2,3]
```

```
% hCriteria: hidden neurones criteria.
% repetitions

function[pointsTable] = generatePointsTable(genesTable,resultsTable,allG,pCriteria
      ,hCriteria,repetitions);

pointsTable = genesTable;

for GjIdx=1:size(allG,2)
     Gj = allG(GjIdx);
     restG = setdiff(allG,Gj);

     % obtains the row indexs where column J has the value Gj
     colJesIdx = find(resultsTable(:,2)==Gj);

     % obtains row indexs which matches with selected H values.
     evalStr=['colHesIdx = find(resultsTable(:,6)',hCriteria,');'];
     eval(evalStr);

     % obtains row index to be evaluated
     filterIdx = intersect(colJesIdx,colHesIdx);

     pointsRow = [];
     for r=1:repetitions
         colRepIdx = find(resultsTable(:,5)== r);
         finalIdx = intersect(filterIdx,colRepIdx);
         subTable =  sortrows(resultsTable(finalIdx,:),7);
         candidates = subTable(:,1);
         [valCand,idxCand] = unique(candidates,'first');
         orderCand = candidates(sort(idxCand));
         switch pCriteria
            case 1,  % assigns one point to both first different genes
                pointsRow = [pointsRow,orderCand(1),orderCand(2)];
            case 2, % assigns one point to both first genes (they can be the same)
                pointsRow = [pointsRow,candidates(1),candidates(2)];
            case 3, % assigns two points to the first gene and one point to the
                 second
                pointsRow = [pointsRow,orderCand(1),orderCand(1),orderCand(2)];
         end
     end

     iElements = unique(pointsRow);
     for iE = 1:length(iElements)
           GiIdx = find(allG==iElements(iE));
           pointsTable(GiIdx,GjIdx) = length(find(pointsRow==iElements(iE)));
     end
end
```

Listado B.11: Módulo Generating Ranking Table

```
% Generate the ranking criteria table.
```

```
2 %
3 % Mariano Rubiolo
4 %
5 % Input parameters...
6 % genesTable: table of genes relations.
7 % resultsTable: table of results.
8 % allG: list of genes.
9 % N: number of elements in the ranking
10 % hCriteria: hidden neurones criteria.
11 % repetitions
12
13 function[rankingTable] = generateRankingTable(genesTable,resultsTable,allG,N,
      hCriteria,repetitions);
14
15 rankingTable = genesTable;
16
17 for GjIdx=1:size(allG,2)
18      Gj = allG(GjIdx);
19      restG = setdiff(allG,Gj);
20
21      % obtains the row indexs where column J has the value Gj
22      colJesIdx = find(resultsTable(:,2)==Gj);
23
24      % obtains row indexs which matches with selected H values.
25      evalStr=['colHesIdx = find(resultsTable(:,6)',hCriteria,');'];
26      eval(evalStr);
27
28      % obtains row indexs to be evaluated
29      finalIdx = intersect(colJesIdx,colHesIdx);
30
31
32      subTable = resultsTable(finalIdx,:);
33      rankingRow = [];
34      for n=1:N
35          rankingRow = [rankingRow,subTable(n,1)];
36      end
37
38      iElements = unique(rankingRow);
39      for iE = 1:length(iElements)
40            GiIdx = find(allG==iElements(iE));
41            rankingTable(GiIdx,GjIdx) = length(find(rankingRow==iElements(iE)));
42      end
43 end
```

Listado B.12: Módulo Generating Simetric Table

```
1 % Resolve the simetric cases.
2 %
3 % Mariano Rubiolo
4 %
5 % Input parameters...
```

```
% table: table of genes relations values.
% mseTable: contains the minimum or the average MSE value.
% alfa: alfa value [0,...,1]
% repetitions

function[auxTable] = generateSimetricPyETable(table,mseTable,alfa)

auxTable = table;
numElements = size(table,1);

for rIdx=2:numElements
     for cIdx=1:rIdx-1

        if (table(rIdx,cIdx) > 0) && (table(cIdx,rIdx) > 0)
            mseIj = mseTable(rIdx,cIdx);
            mseJi = mseTable(cIdx,rIdx);
            poiIj = table(rIdx,cIdx);
            poiJi = table(cIdx,rIdx);

            dValue = obtainDValue(poiIj,poiJi,mseIj,mseJi,alfa);
            if dValue > 0  % if d es greater than 0, "gene i regulates gene j"
                auxTable(cIdx,rIdx) = 0;
            elseif dValue < 0  % if d is less than 0, "gene j regulates gene i".
                auxTable(rIdx,cIdx) = 0;
            end
        end

     end
end
```

Listado B.13: Módulo Generating Unchained Table

```
% Resolve the chained cases.
%
% Mariano Rubiolo
%
% Input parameters...
% table: table of genes relations values.
%

function[newTable] = generateUnchainedPTable(table)

newTable = table;
numElements = size(table,1);

for cIdx=1:numElements
       for rIdx=1:numElements
       if table(rIdx,cIdx) > 0

                % obtaining the pre-regulators of cIdx
               preRegIdxs = obtainPreRegulators(table,cIdx);
```

```
20
21                if size(preRegIdxs,1) > 0
22
23                    for pr=1:size(preRegIdxs,1)
24                        if table(preRegIdxs(pr,1),cIdx) < table(preRegIdxs(pr,2),
                              cIdx)
25                         newTable(preRegIdxs(pr,1),cIdx) = 0;
26                        end
27                    end
28                end
29            end
30        end
31    end
32 end
```

Listado B.14: Módulo Generating Threshold Table

```
1 % Resolves thresholds cases.
2 % (Those less than point threshold. Those greater than MSE threshold).
3 %
4 % Mariano Rubiolo
5 %
6 % Input parameters...
7 % table: table of genes relations values.
8 % mseTable: table of min MSE for each gene relations.
9 % percentagePOI: threshold for Points.
10 % percentageMSE: threshold for MSE.
11 % condition: application of one or both of the thresholds ('and'/'or').
12
13
14 function[newTable] = generateThresholdPyETable(table,mseTable,percentagePOI,
       percentageMSE,condition)
15
16 newTable = table;
17
18 numElements = size(table,1);
19
20 for cIdx=1:numElements
21
22     columnPoi = table(:,cIdx);
23     columnMse = mseTable(:,cIdx);
24
25     % Normalization
26     colPoiNorm = mapminmax(columnPoi',0,1);
27     colMseNorm = mapminmax(columnMse',0,1);
28
29     [maxPoiVal, idxMaxVal] = max(colPoiNorm);
30     [minMSEVal, idxMinVal] = min(columnMse(find(columnMse~=0)));
31
32     thresholdPOI = 100-percentagePOI/100;
33     thresholdMSE = percentageMSE/100;
```

```
34
35
36 condition = 'and';
37 switch condition
38         case('and')  % Both threshold conditions must be met.
39             for rIdx=1:numElements
40                 if (colPoiNorm(rIdx) <= thresholdPOI) && (colMseNorm(rIdx) >=
                       thresholdMSE)
41                     newTable(rIdx,cIdx) = 0;
42                 end
43             end
44         case('or')  % One of the threshold conditions must be met.
45             for rIdx=1:numElements
46                 if (colPoiNorm(rIdx) <= thresholdPOI) | (colMseNorm(rIdx) >=
                       thresholdMSE)
47                     newTable(rIdx,cIdx) = 0;
48                 end
49             end
50        end
51 end
```

Listado B.15: Módulo Obtaining PreRegulators

```
1 % Obtains the list of genes which pre-regulates the gen in cIdx.
2 %
3 % Mariano Rubiolo
4 %
5 % Input parameters...
6 % table: table of genes relations values.
7 % cIdx: index of the gene whose pre-regulators we are looking for.
8
9 function[preRegIdxs] = obtainPreRegulators(table,cIdx)
10
11 preRegIdxs = [];
12 % finding the existing genes relations
13 pRlist = find(table(:,cIdx)');
14 if size(pRlist,2) > 0
15    for pr=1:size(pRlist,2)
16
17        pRlist2 = find(table(:,pRlist(pr))');
18        if size(pRlist2,2) ~= 0
19             % finding the pre-regulators of the pre-regulator
20             for pr2 = 1:size(pRlist2,2)
21             exist = find(pRlist==pRlist2(pr2));
22             if size(exist,2) ~= 0
23                 preRegIdxs = [preRegIdxs; [pRlist2(pr2) pRlist(pr) cIdx]];
24             end
25             end
26
27         end
28     end
```

```
29 else
30     preRegIdxs = [];
31 end
```

Bibliografía

Aitkenhead, M. J. y McDonald, A. J. S. (2003). «A neural network face recognition system». *Engineering Applications of Artificial Intelligence*, **16(3)**, pp. 167 – 176.

Ando, S. y Iba, H. (2001). «Inference of gene regulatory model by genetic algorithms». En: *Evolutionary Computation, 2001. Proceedings of the 2001 Congress on*, volumen 1, pp. 712–719 vol. 1.

Ao, S. I. y Palade, V. (2011). «Ensemble of Elman neural networks and support vector machines for reverse engineering of gene regulatory networks». *Appl. Soft Comput.*, **11(2)**, pp. 1718–1726. ISSN 1568-4946.

Badica, C.; Mangioni, G. y Rahimi, S. (2010). «Intelligent distributed information systems». *Information Sciences - Special Issue on Intelligent Distributed Information Systems*, **180(10)**, pp. 1779–1780.

Barker, Nathan A.; Myers, Chris J. y Kuwahara, Hiroyuki (2011). «Learning Genetic Regulatory Network Connectivity from Time Series Data». *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, **8(1)**, pp. 152–165. ISSN 1545-5963.

Bengio, Yoshua (2009). «Learning Deep Architectures for AI». *Found. Trends Mach. Learn.*, **2(1)**, pp. 1–127. ISSN 1935-8237.

Berlin, J. y Motro, A. (2002). «Database schema matching using machine learning with feature selection». 1, pp. 452–466.

Bianchini, M.; Maggini, M.; Sarti, L. y Scarselli, F. (2005). «Recursive neural networks learn to localize faces». *Pattern Recognition Letters*, **26(12)**, pp. 1885 – 1895.

Bilke, Alexander y Naumann, Felix (2005). «Schema matching using duplicates». En: *Data Engineering, 2005. ICDE 2005. Proceedings. 21st International Conference on*, pp. 69–80. IEEE.

Bishop, Christopher M. (1995). *Neural Networks for Pattern Recognition*. Oxford University Press, Inc., New York, NY, USA. ISBN 0198538642.

Bornholdt, S. (2008). «Boolean network models of cellular regulation: prospects and limitations.» *J R Soc Interface*, **5 Suppl 1**.

Brío, B.M. y Molina, A.S. (2007). *Redes neuronales y sistemas borrosos*. Alfaomega. ISBN 9789701512500.

Buciluǎ, Cristian; Caruana, Rich y Niculescu-Mizil, Alexandru (2006a). «Model compression». En: *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 535–541. ACM, New York, NY, USA. ISBN 1-59593-339-5.

Buciluǎ, Cristian; Caruana, Rich y Niculescu-Mizil, Alexandru (2006b). «Model compression». En: *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 535–541. ACM, New York, NY, USA. ISBN 1-59593-339-5.

Burrascano, P.; Fiori, S. y Mongiardo, M. (1999). «A review of artificial neural networks applications in microwave computer-aided design». *International Journal of RF and Microwave Computer-Aided Engineering*, **9(3)**, pp. 158–174.

Capello, D.; Martinez, C.; Milone, D.H. y Stegmayer, G. (2009). «Array of Multilayer Perceptrons with No-class Resampling Training fr Face Recognition». *Revista Iberoamericana de Inteligencia Artificial*, **13(44)**, pp. 5–13.

Castano, S.; Ferrara, A. y Montanelli, S. (2006). «Matching Ontologies in Open Networked Systems: Techniques and Applications». *Journal on Data Semantics (JoDS)*, **3680(5)**, pp. 25–63.

Chaturvedi, D.K. (2008). *Soft Computing Techniques - Studies in Computational Intelligence*. Springer-Verlag, London.

Chortaras, A.; Stamou, G.B. y Stafylopatis, A. (2005). «Learning Ontology Alignments Using Recursive Neural Networks». 1, pp. 811–816.

Chowdhury, Ahsan; Chetty, Madhu y Vinh, Nguyen (2013). «Incorporating time-delays in S-System model for reverse engineering genetic networks». *BMC Bioinformatics*, **14(1)**, p. 196. ISSN 1471-2105.

Cun, Yann Le; Denker, John S. y Solla, Sara A. (1990). «Optimal Brain Damage». En: *Advances in Neural Information Processing Systems*, pp. 598–605. Morgan Kaufmann.

Curino, C.; Orsi, G. y Tanca, L. (2007). «X-SOM: A Flexible Ontology Mapper». volumen 1, pp. 424–428.

Cybenko, G. (1989). «Approximation by superpositions of sigmoidal functions». *Math. Control, Sygnals, Syst.*, **2**, pp. 303–314.

Davies, J.; Studer, R. y Warren, P. (2007). *Semantic Web Technologies: trends and research in ontology-based systems*. John Wiley, London.

Doan, A.; Domingos, P. y Halevy, A. (2003). «Learning to match the schemas of data sources: A multistrategy approach». *Machine Learning*, **50(3)**, pp. 279–301.

Doan, A. y Halevy, A. Y. (2005). «Semantic Integration Research in the Database Community: A Brief Survey». *AI Magazine*, **26(1)**, pp. 83–94.

Doan, A.; Madhavan, J.; Domingos, P. y Halevy, A. (2004). *Ontology Matching: A Machine Learning Approach. Handbook on Ontologies in Information Systems*. Springer, New York.

Duda, R. O. y Hart, P. E. (2003). *Pattern Classification and Scene Analysis*. Wiley Interscience, second[a] edición.

Dzeroski, S. y Zenko, B. (2004). «Is Combining Classifiers with Stacking Better than Selecting the Best One?» *Machine Learning*, **54**, pp. 255–273.

Ehrig, M. (2007). *Ontology Alignment: Bridging the Semantic Gap*. Springer, London.

Euzenat, J. y Shvaiko, P. (2007). *Ontology Matching*. Springer, London.

Gomez-Perez, A.; Corcho, O. y Fernandez-Lopez, M. (2005). *Ontological Engineering*. Springer, London.

Hache, Hendrik; Lehrach, Hans y Herwig, Ralf (2009). «Reverse Engineering of Gene Regulatory Networks: A Comparative Study». *EURASIP Journal on Bioinformatics and Systems Biology*, **2009(1)**, p. 617281. ISSN 1687-4153.

Hartemink, A. J. (2005). «Reverse engineering gene regulatory networks». *Nat Biotech*, **23(5)**, pp. 554–555. ISSN 1087-0156.

Hassibi, Babak; Stork, David G. y Com, Stork Crc. Ricoh. (1993). «Second Order Derivatives for Network Pruning: Optimal Brain Surgeon». En: *Advances in Neural Information Processing Systems 5*, pp. 164–171. Morgan Kaufmann.

Haykin, S. (2005). *Neural Networks*. MacMillan, New York.

Haykin, S. (2007). *Neural Networks: A Comprehensive Foundation (3rd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA. ISBN 0131471392.

Haykin, Simon (2008). *Neural Networks and Learning Machines (3rd Edition)*. Prentice Hall, 3ª edición. ISBN 0131471392.

Hecker, Michael; Lambeck, Sandro; Toepfer, Susanne; van Someren, Eugene y Guthke, Reinhard (2009). «Gene regulatory network inference: Data integration in dynamic models. A review». *Biosystems*, **96(1)**, pp. 86–103. ISSN 03032647.

Higa, Carlos H. A.; Andrade, Tales P. y Hashimoto, Ronaldo F. (2013). «Growing Seed Genes from Time Series Data and Thresholded Boolean Networks with Perturbation». *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, **10(1)**, pp. 37–49. ISSN 1545-5963.

Hornik, K.; Sthincombe, M. y White, H. (1989). «Multilayer feedforward networks are universal approximators». *Neural Networks*, **2(1)**, pp. 359–366.

Huang, J.; Dang, J.; Zheng, W.J. y Huhns, M. (2008). «Use Artificial Neural Network to Align Biological Ontologies». *BMC Genomics*, **9(16)**, pp. 1–12.

Husmeier, Dirk (2003). «Sensitivity and specificity of inferring genetic regulatory interactions from microarray experiments with dynamic Bayesian networks». *Bioinformatics*, **19(17)**, pp. 2271–2282.

Ibrahim, Zina M.; Ngom, Alioune y Tawfik, Ahmed Y. (2011). «Using Qualitative Probability in Reverse-Engineering Gene Regulatory Networks». *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, **8(2)**, pp. 326–334. ISSN 1545-5963.

Jackson, J.E. (1991). *A user's guide to principal components*. Wiley series in probability and mathematical statistics: Applied probability and statistics. Wiley.

Jong, Hidde De (2002). «Modeling and simulation of genetic regulatory systems: A literature review». *Journal of Computational Biology*, **9**, pp. 67–103.

Jung, J. (2010). «Reusing ontology mappings for query routing in semantic peer-to-peer environment». *Information Sciences*, **180(17)**, pp. 3248–3257.

Kavzoglu, Taskin y Mather, Paul M. (1998). «Assessing Artificial Neural Network Pruning Algorithms». En: *Proceedings of the 24 th Annual Conference and Exhibition of the Remote Sensing Society*, pp. 603–609.

Kibangou, A. y Favier, Gerard (2009). «Identification of fifth-order Volterra systems using i.i.d. inputs.» *IET Signal Processing*, **4**, pp. 30–44.

Kirby, M. y Sirovich, L. (1990). «Application of the Karhunen-Loeve Procedure for the Characterization of Human Faces». *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **12(1)**, pp. 103–108.

Knott, S.; Mostafavi, S. y Mousavi, P. (2010). «A neural network based modeling and validation approach for identifying gene regulatory networks». *Neurocomputing*, **73(13-15)**, pp. 2419–2429.

Kong, S.; Heo, J.; Abidi, B.; Palk, J. y Abidi, M. (2005). «Recent advances in visual and infrared face recognition-a review». *Computer Vision and Image Understanding*, **97(1)**, pp. 103–135.

Korenberg M. J., Ian W. Hunterb, Davidb R. y Solomonc, Jerry E. (2001). «Parallel cascade identification and its application to protein family prediction». *Journal of Biotechnology*, **91**, pp. 35–47.

Li, Stan y Jain, Anil (Eds.) (2004). *Handbook of Face Recognition*. Springer-Verlag.

Li, W. y Clifton, C. (1995). «Semint: A System Prototype for Semantic Integration in Heterogeneous Databases». 1, pp. 484–485.

Liu, Guixia; Liu, Lei; Liu, Chunyu; Zheng, Ming; Su, Lanying y Zhou, Chunguang (2011). «Combination of Neuro-Fuzzy Network Models with Biological Knowledge for Reconstructing Gene Regulatory Networks». *Journal of Bionic Engineering*, **8(1)**, pp. 98 – 106. ISSN 1672-6529.

Mao, Ming; Peng, Yefei y Spring, Michael (2010). «An adaptive ontology mapping approach with neural network based constraint satisfaction». *Journal of Web Semantics*, **8(1)**, pp. 14–25.

Marquardt, Donald W. (1963). «An Algorithm for Least-Squares Estimation of Nonlinear Parameters». *SIAM Journal on Applied Mathematics*, **11(2)**, pp. 431–441.

Martinez, A. y Kak, A. (2001). «PCA versus LDA». *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **23(2)**, pp. 228–233.

Meireles, M. R. G.; Almeida, P. E. M. y Simoes, M. G. (2003). «A comprehensive review for industrial applicability of artificial neural networks». *Industrial Electronics, IEEE Transactions on*, **50(3)**, pp. 585–601.

Mitra, Sushmita; Das, Ranajit y Hayashi, Yoichi (2011). «Genetic Networks and Soft Computing». *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, **8(1)**, pp. 94–107. ISSN 1545-5963.

Muraro, D.; Voss, U.; Wilson, M.; Bennett, M.; Byrne, H.; Smet, I. De; Hodgman, C. y King, J. (2013). «Inference of the Genetic Network Regulating Lateral Root Initiation in Arabidopsis thaliana». *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, **10(1)**, pp. 50–60. ISSN 1545-5963.

Nam, Sang-Won y Powers, E.J. (2003). «Volterra series representation of time-frequency distributions». *Signal Processing, IEEE Transactions on*, **51(6)**, pp. 1532–1537. ISSN 1053-587X.

Nguyen, Derrick y Widrow, Bernard (1990). «Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights.» En: *Proceedings of the International Joint Conference on Neural Networks*, pp. 21–26.

Nounou, M.; Serpedin, E.; Noor, A. y Nounou, Hazem N. (2012). «Inferring Gene Regulatory Networks via Nonlinear State-Space Models and Exploiting Sparsity». *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, **9(4)**, pp. 1203–1211. ISSN 1545-5963.

Olson, D. y Delen, D. (2008). *Advanced Data Mining*. Springer.

Pedro, José C.; Madaleno, Joana C. y García, José A. (2003). «Theoretical basis for extraction of mildly nonlinear behavioral models». *International Journal of RF and Microwave Computer-Aided Engineering*, **13(1)**, pp. 40–53. ISSN 1099-047X.

Phillips, P. Jonathon; Moon, Hyeonjoon; Rizvi, Syed A. y Rauss, Patrick J. (2000). «The FERET evaluation methodology for face-recognition algorithms». *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **22**, pp. 1090–1104.

Pinna, Andrea; Heise, Sandra; Flassig, Robert; de la Fuente, Alberto y Klamt, Steffen (2013). «Reconstruction of large-scale regulatory networks based on perturbation graphs and transitive reduction: improved methods and their evaluation». *BMC Systems Biology*, **7(1)**, p. 73. ISSN 1752-0509.

Quadrana, Leandro; Almeida, Juliana; Otaiza, Santiago N.; Duffy, Tomas; CorrÃªa da Silva, JuniaV.; Godoy, Fabiana; AsÃs, Ramon; BermÃºdez, Luisa; Fernie, AlisdairR.; Carrari, Fernando y Rossi, Magdalena (2013). «Transcriptional regulation of tocopherol biosynthesis in tomato». *Plant Molecular Biology*, **81(3)**, pp. 309–325. ISSN 0167-4412.

Rahman, Ashfaqur y Verma, Brijesh (2011). «Novel Layered Clustering-Based Approach for Generating Ensemble of Classifiers». *IEEE Transactions on Neural Networks*, **22(5)**, pp. 781–792.

Ram, Ramesh y Chetty, Madhu (2011). «A Markov-Blanket-Based Model for Gene Regulatory Network Inference». *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, **8(2)**, pp. 353–367. ISSN 1545-5963.

Rubiolo, M. (2012). «Extended evaluation of the Volterra-Neural Network for model compression». En: *13th Argentine Symposium on Artificial Intelligence (ASAI 2012) on 41th Argentine Conference on Informatics*, ISSN 1850-2784.

Rubiolo, M.; Caliusco, M.L.; Stegmayer, G.; Coronel, M. y Fabrizi, M. Gareli (2012). «Knowledge discovery through ontology matching: An approach based on an Artificial Neural Network model.» *Information Sciences*, **194**, pp. 107–119. ISSN 0020-0255.

Rubiolo, M.; Caliusco, M.L.; Stegmayer, G.; Gareli, M. y Coronel, M. (2009). «Knowledge Source Discovery: An experience using Ontologies, WordNet and Artificial Neural Networks». En: *13th International Conference on Knowledge-Based and Intelligent Information & Engineering Systems (KES)*, pp. 11–17.

Rubiolo, M.; Stegmayer, G. y Milone, D. (2010). «Compressing a Neural Network Classifier using a Volterra-Neural Network model». En: *IEEE International Joint Conference on Neural Networks (IJCNN)*, pp. 1–7.

Rubiolo, M.; Stegmayer, G. y Milone, D. (2013a). «A novel approach for gene regulatory network mining from temporal series data using neural networks». *IEEE Transactions on Computational Biology and Bioinformatics. (En revisión).*

Rubiolo, M.; Stegmayer, G. y Milone, D. H. (2013b). «Compressing arrays of classifiers using Volterra-Neural Network: application to face recognition». *Neural Computing & Applications*, **23(6)**, pp. 1687–1701.

Rumelhart, D.E.; Hinton, G.E. y Williams, R. J. (1986). «Learning representations by back-propagating errors». *Nature*, **323(1)**, pp. 533–536.

Salomon, David (2007). *Data Compression: The Complete Reference*. Springer.

Shimshoni, Yair y Intrator, Nathan (1996). «Classification of Seismic Signals by Integrating Ensembles of Neural Networks». *IEEE Transactions on Signal Processing*, **46**, pp. 1194–1201.

Siebel, Nils T.; Bötel, Jonas y Sommer, Gerald (2009). «Efficient neural network pruning during neuro-evolution». En: *IJCNN'09: Proceedings of the 2009 international joint conference on Neural Networks*, pp. 3147–3154. IEEE Press.

Smith, Barry; Kusnierczyk, Waclaw; Schober, Daniel y Ceusters, Werner (2006). «Towards a Reference Terminology for Ontology Research and Development in the Biomedical Domain.» En: *KR-MED*, volumen 222.

Smith, M.; Welty, C. y McGuiness, D. (2004). «OWL web ontology language guide». *Recommendation W3C*, **2(1)**.

Smith, V. A.; Jarvis, E. D. y Hartemink, A. J. (2002). «Evaluating functional network inference using simulations of complex biological systems». *Bioinformatics*, **18(1)**, pp. S216–S224.

Staab, Steffen y Studer, Rudi (2009). *Handbook on Ontologies*. Springer Publishing Company, Incorporated, 2nd[a] edición. ISBN 3540709991, 9783540709992.

Stegmayer, G.; Caliusco, M.L.; Chiotti, O. y Galli, M.R. (2007). «NN-agent for distributed Knowledge Source Discovery». *Lecture Notes on Computer Science 4805*, **1(1)**, pp. 467–476.

Stegmayer, G. y Chiotti, O. (2009). «Volterra NN-based behavioral model for new wireless communications devices». *Neural Computing and Applications*, **18**, pp. 283–291.

Stumme, G. y Maedche, A. (2001). «FCA-Merge: Bottom-up merging of ontologies». pp. 225–234.

Styczynski, Mark P. y Stephanopoulos, Gregory (2005). «Overview of computational methods for the inference of gene regulatory networks». *Computers & Chemical Engineering*, **29(3)**, pp. 519 – 534.

Tasoulis, D.K.; Plagianakos, V.P. y Vrahatis, M.N. (2008). *Computational Intelligence in Bioinformatics*. volumen 94 de *Studies in Computational Intelligence*. Springer.

Turk, Matthew y Pentland, Alex (1991). «Eigenfaces for recognition». *J. Cognitive Neuroscience*, **3**, pp. 71–86. ISSN 0898-929X.

Volterra, V. (1959). *Theory of Functionals and Integral and Integro-Differential Equations*. Dover.

Wang, L. y Fu, X. (2005). *Data Mining with Computational Intelligence*. Advanced Information and Knowledge Processing Series. Springer-Verlag Berlin and Heidelberg GmbH & Company KG.

Wang, Rui-Sheng; Wang, Yong; Zhang, Xiang-Sun y Chen, Luonan (2007). «Inferring transcriptional regulatory networks from high-throughput data». *Bioinformatics*, **23(22)**, pp. 3056–3064. ISSN 1367-4803.

Werhli, A. y Husmeier, D. (2007). «Reconstructing Gene Regulatory Networks with Bayesian Networks by Combining Expression Data with Multiple Sources of Prior Knowledge». *Statistical Applications in Genetics and Molecular Biology*, **6(1)**.

Widrow, B. y Lehr, M. A. (2002). «30 years of adaptive neural networks: perceptron, Madaline, and backpropagation». *Proceedings of the IEEE*, **78(9)**, pp. 1415–1442.

Witten, I.H. y Frank, E. (2005). *Data Mining: Practical Machine Learning Tools and Techniques, Second Edition*. The Morgan Kaufmann Series in Data Management Systems. Elsevier Science. ISBN 9780080477022.

Xu, Rui y Wunsch, Donald C. (2009). *Clustering*. Wiley and IEEE Press.

Yang, Bin; Chen, Yuehui y Jiang, Mingyan (2013). «Reverse engineering of gene regulatory networks using flexible neural tree models». *Neurocomput.*, **99**, pp. 458–466. ISSN 0925-2312.

Yu, Dong y Deng, Li (2011). «Deep Learning and Its Applications to Signal and Information Processing [Exploratory DSP]». *Signal Processing Magazine, IEEE*, **28(1)**, pp. 145–154.

Yu, Dong; Seltzer, Michael L.; Li, Jinyu; Huang, Jui-Ting y Seide, Frank (2013). «Feature Learning in Deep Neural Networks - A Study on Speech Recognition Tasks». *CoRR*, **abs/1301.3605**, pp. 1–9.

Yu, J.; Smith, V. A.; Wang, P. P.; Hartemink, A. J. y Jarvis, E. D. (2004). «Advances to Bayesian network inference for generating causal networks from observational biological data». *Bioinformatics*, **20(18)**, pp. 3594–3603.

Zhang, D. y Wangmeng, Z. (2007). «Computational Intelligence-Based Biometric Technologies». *IEEE Computational Intelligence Magazine*, **2(2)**, pp. 26–36.

Zhao, W.; Chellappa, R.; Phillips, P. J. y Rosenfeld, A. (2003). «Face recognition: A literature survey». *ACM Computing Surveys*, **35(4)**, pp. 399–458.