MODELO DE PREDICCIÓN DE DAÑOS, UNA APLICACIÓN DE REDES NEURONALES ARTIFICIALES A RIESGOS EN EL SEGURO DE NARANJA

PRESENTADA POR:

CATALINA LOZANO MURCIA

CO TUTORES:

JOSE MIGUEL RODRIGUEZ PARDO DEL CASTILLO

JESUS RAMÓN SIMÓN DEL POTRO

JUNIO DE 2015

Universidad
Carlos III de Madrid

AGRADECIMIENTOS

Gracias a Dios por brindarme esta gran oportunidad en la vida, a Jose Miguel, por ser algo más que un tutor, demostrando su cercanía en estos meses de duro trabajo y dificultades. Agroseguro, en especial a Pablo por su confianza y Margarita por su apoyo e interés incondicional, y finalmente a mi familia, y aquellos quienes en los últimos meses se han convertido ella, haciendo mi vida en un país ajeno mucho más amena, GRACIAS.

CONTENIDO

I. INTRODUCCIÓN

En el desarrollo de programas informáticos que permitan el abordaje de temas más compuestos, la ciencia ha tornado su visión de nuevo a la naturaleza y la biología como ejemplo a seguir. En los inicios del método científico todo se basaba en la observación, hasta hace poco este sistema parecía alejarse para ocuparse solo de construcciones abstractas, en ocasiones, muy apartadas de la realidad aunque teóricamente robustas, que no logran modelar problemas reales y si se embarcan en construcciones complejas que solo parecieran lucir una alta capacidad en programación.

Contrario a esto, hay una tendencia enfocada a la retoma de los inicios científicos, donde se aprecia el valor de la naturaleza como gestora de procesos y modelos que en ella están intrínsecos, pero que tienen las claves para el desarrollo de nuevos patrones de investigación. Del mismo modo que las líneas del diseño y la publicidad están revolucionándose con el *Neuromarketing*, la medicina alternativa compite con la *medicina regenerativa*, cada vez se desarrollan más teorías a partir de la *Neuroeconomía,* así como se combina la psicología, la pedagogía, y la neurología para dar paso al *Neuroaprendizaje*; la ciencia actuarial se dirige también a dichos enfoques, para dar respuesta a sus problemas básicos a través de esquemas basados en procesos naturales.

El siglo XXI es el siglo del hemisferio derecho; luego de años de predominio del hemisferio izquierdo con la racionalidad, las matemáticas y la lógica, se da paso a un ciclo preocupado por las emociones, los sentimientos, la creatividad y el arte. Los esfuerzos científicos en las distintas aéreas se dirigen a entender el comportamiento humano, su manera de pensar, sus gustos, sus patrones como un factor diferenciador y determinante, y es ahí donde los procesos biológicos cobran importancia, pues son los que señalan el camino a propuestas de nuevos desarrollos y enfoques.

I.1. MOTIVACIÓN

Las ciencias parten de fundamentos de procesos computacionales para el desarrollo de la inteligencia artificial, modelando sistemas que lograran aprender, generar patrones y predecir. Es el enfoque en la ciencia artificial la que en la actualidad da respuestas en modelos de predicción de insolvencia, detección de fraude, modelos de clasificación de riesgo, análisis de la demanda, entre otros.

Basta con observar las potencialidades de esta rama, para darse cuenta la gran motivación que existe en el estudio de los sistemas de autoaprendizaje y su aplicación en las finanzas y ciencias actuariales, pues son un recurso innovador que va más allá del conocimiento convencional para entender el comportamiento de los datos y transformarlos en información, superando las barreras impuestas por las hipótesis inflexibles de los métodos estadísticos tradicionales.

Para aprovechar las ventajas del símil entre el sistema neuronal y los procesos informáticos, se ha desarrollado la técnica de procesamiento denominada Redes Neuronales Artificiales (RNA), como una herramienta estadística para el procesamiento de información a través de unidades adaptativas y con alta interconexión, siendo muy similares al proceso biológico neuronal del sistema nervioso y ofreciendo una gran variedad de usos.

I.2. OBJETIVOS

En el marco de las necesidades actuales en el mundo de los seguros y las finanzas, donde el tratamiento de grandes bases de datos y la generación de modelos que rompan con los esquemas y los limitantes tradicionales para el entendimiento y gestión de riesgos, se pretende en este trabajo

establecer un modelo que permita valorar los daños a causa de los riesgos de Viento y Helada en Ribera del Jucar para el seguro de cultivo de naranja, realizando un pequeño acercamiento a los modelos de Redes Neuronales Artificiales aplicadas a riesgos en el sector asegurador agrícola, con el fin de generar un modelo alternativo de estimación de daños por póliza que sirva como base de estimación de las pérdidas esperadas agregadas y generación de tarifas para cada póliza, empleando solamente la información histórica del producto que se desea evaluar.

Como parte del desarrollo del trabajo se pretende realizar una aplicación con datos reales, para lo cual y gracias a la colaboración prestada por parte de Agroseguro S.A., entidad gestora de los seguros agrarios en España, quienes facilitaron los datos y la información referente a la estructura y comportamiento del seguro analizado, se eligió el seguro de Naranja, por ser uno de los productos insignia de la compañía, que cuenta con información histórica sufieciente y robusta, para el ámbito geográfico con mayor contratación histórica (Ribera del Jucar), sirviendo sus resultados para el análisis del producto en este municipio y ofreciendo una alternativa a la medición de los daños esperados, contrastable con la situación actual del seguro.

I.3. DESCRIPCIÓN DEL TRABAJO

En las siguientes páginas se expone el desarrollo de un modelo que cuenta con dos Redes Neuronales Artificiales (RNA) aplicados a los riesgos de Helada y Viento para el seguro de Naranja en la comarca Ribera de Jucar, yendo desde la historia y las generalidades de los modelos de esta índole, hasta los detalles de entrenamiento, estimaciones, resultados y conclusiones.

A demás se propone una solución iterativa al problema de inestabilidad en la generación de pesos y diseño óptimo de RNAs y se supera, en cierta medida, el enfoque de caja negra que históricamente se le ha dado a estos

modelos, mediante el uso de un indicador de importancia de las variables dentro de la determinación de la red, de esta manera se consigue además del resultado objetivo una herramienta que puede llegar a tener un enfoque comercial para las políticas de diseño, gestión y promoción del seguro.

I.4. RESUMEN DE LOS RESULTADOS

En general los resultados reflejan modelos de gran ajuste y poder predictivo, tanto en el estimación por póliza como en el agregado, sin embargo es de destacar la difícil tarea que supone el diseño y optimización de los modelos RNA, demandando un coste computacional considerable y dando pie al desarrollo de trabajos posteriores sobre modelos de optimización de RNA para aplicaciones actuariales.

Los resultados del trabajo ofrecen una nueva ventana al ajuste de modelos alternativos para los problemas actuariales básicos, además de ser herramientas de un potencial muy grande que no solo se limitan a la estimación de valores si no que además, dentro del diseño, desarrollo y optimización de los modelos el diseñador puede re evaluar el estado real del problema, su composición y la influencia de las diferentes variables en el sistema. Por otro lado se hace un llamamiento a la academia al desarrollo de modelos mucho más completos, que superen las barreras de diseño mediante la generación de estudios de optmización en el definición de arquitectura y topología de las RNA enfocadas a los problemas de esta rama.

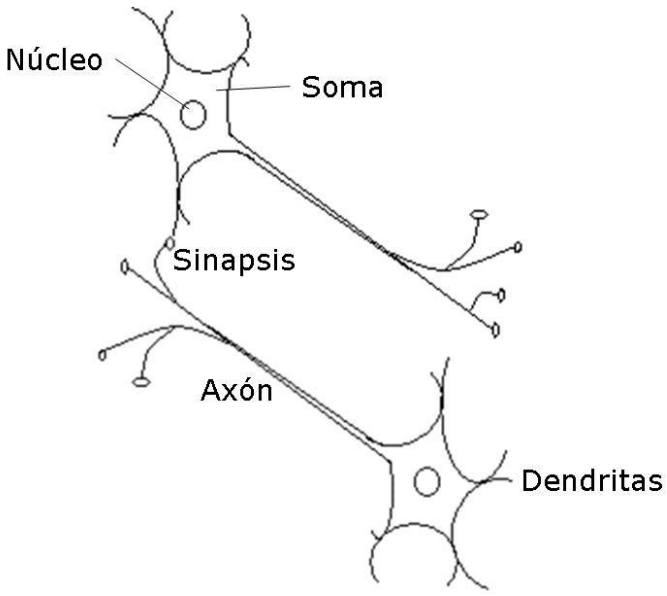
II. REVISIÓN DE LA LITERATURA

Para el desarrollo del presente trabajo se realizó una revisión bibliográfica de los principales trabajos prácticos en términos de aplicaciones de modelos de Redes Neuronales Artificiales y su base teórica, remitiéndose de esta manera hasta una contextualización desde el punto de vista biológico; a continuación se presentará un corto resumen de manera contraria partiendo de la definición biológica, pasando por la conslidación teórica de los modelos, su evolución histórica y finalmente se exponen algunas aplicaciones financiero actuariales.

II.1. NEURONAS

El cerebro está compuesto por una red de billones de neuronas, con cerca de 100.000 conexiones por elemento, cada neurona es una unidad procesadora que recibe señales, las combina y según sea el caso genera una respuesta de salida. Como muestra la Ilustración 1 las neuronas tienen tres componentes principales: las dendritas, el cuerpo de la célula o soma, y el axón.

Ilustración 1. Partes de una neurona



Fuente: Elaboración propia

El soma realiza la suma de las señales de entrada, el axón que es una fibra larga, lleva la señal desde el cuerpo hacia otras neuronas y finalmente el punto de contacto entre un axón y una dendrita de otra neurona es la sinapsis. Su longitud es determinada por la complejidad del proceso químico que estabiliza la función de la red neuronal. Todas las neuronas conducen la información de forma similar, a través de los axones mediante breves impulsos eléctricos, denominados potenciales de acción.

Los potenciales de acción, son señales de baja frecuencia conducidas lentamente por neurotransmisores (transmisores químicos que son liberados en las sinapsis). La neuronas reciben las entradas procedentes de sinapsis excitadoras, cuyos neurotransmisores provocan disminuciones de potencial en la membrana de la célula pos-sináptica, facilitando la generación de impulsos a mayor velocidad, e inhibidoras, cuyos neurotransmisores tienden a estabilizar el potencial de la membrana, dificultando la emisión de impulsos, es decir recibe impulsos amplificados o atenuados. La suma de estos impulsos en el soma determina si será o no estimulada, dependiendo de si el valor de la suma es superior al valor de umbral del potencial de acción (Purves, Fitzpatrick, Hall, Mc Namara, & Williams, 2007).

II.2. REDES NEURONALES ARTIFICIALES (RNA)

La definición de red neuronal artificial (RNA) varía según el autor, desde modelo matemático, herramienta estadística, arreglo de procesamiento computacional, hasta sistema de aprendizaje adaptativo, como se muestra a continuación.

"Una Red neuronal es un modelo computacional, paralelo, compuesto de unidades procesadoras adaptativas con una alta interconexión en ellas" (Hassoun, 1995).

"Sistemas de procesamiento de la información que hacen uso de algunos de los principios que organizan la estructura del cerebro" (Lin & Lee, 1996)

"Sistemas de procesamiento de información que tienen características de funcionamiento comunes con las redes neuronales biológicas" (Fausett, 1994).

"Sistema caracterizado por una red adaptativa combinada con técnicas de procesamiento de información paralelo" (Kung, 1993).

"Desde la perspectiva del reconocimiento de patrones las redes neuronales son una extensión de métodos de clásicos estadísticos" (Bishop, 1995).

Sin embargo para efectos de este trabajo se entenderá como un sistema inteligente de aprendizaje adaptativo, basado en el comportamiento de las neuronas biológicas. Su alcance para solucionar problemas lineales y no lineales a partir del aprendizaje, orientado por información, ofrece una gran flexibilidad a la hora de su implementación.

II.3. RESEÑA HISTÓRICA

Dado el origen biológico del concepto, se puede situar el inicio de la historia de las Redes Neuronales Artificiales con las investigaciones realizadas por Alan Turning, padre de la Inteligencia Artificial, en 1936, con el desarrollo de algoritmos de procesamiento y desifrado, como el empleado por la reconocida máquina enigma.

Los primero estudios sobre el cerebro como referente de análisis computacional fueron propuestos por Alan Turing, pero fue hasta 1943 cuando el neurofisiólogo Warren McCulloch y el matemático Walter Pitts, desarrollaron los fundamentos de la computación neuronal, en su artículo "*A logical calculus of Ideas Imminent in Nervous Activity*" (McCulloch & Pitts, 1943) propusieron una teoría acerca del mecanismo de funcionamiento de las neuronas y desarrollaron una red neuronal simple mediante circuitos

eléctricos. En 1949 Donald Hebb definió el proceso de aprendizaje de una red neuronal mediante reglas de activación que reaccionaban ante cambios.

En el verano de 1956, Minsky, McCarthy, Rochester, Shanon, organizaron el Congreso de Dartmouth, como la primera conferencia sobre Inteligencia Artificial que fue patrocinada por la Fundación Rochester, siendo reconocida como la cuna de la inteligencia artificial.

Para el siguiente año Frank Rosenblatt. Inició el desarrollo de la primera red neuronal que aún hoy es usada, el Perceptron era capaz de aprender y generalizar patrones para luego realizar clasificaciones de información no procesada, sin embargo solo podía solucionar problemas de clasificación lineal y no resolvía el inconveniente de la función "OR exclusiva", posteriormente publica que, bajo ciertas condiciones, el aprendizaje del Perceptron convergía hacia un estado finito.

En 1960 Bernard Widroff y Marcian Hoff desarrollaron la primera red neuronal aplicada a un problema real, el modelo Adaline (ADAptative LINear Elements) fue utilizado comercialmente para generar filtros adaptativos que eliminaban el eco en las llamadas telefónicas, por varias décadas.

Por otro lado Rosemblatt empleo el Perceptron como identificador de patrones ópticos binarios y salidas binarias, generando así la famosa regla de aprendizaje Delta que permite usar señales continuas de entrada y salida.

Pero durante 1969 Minsky y Papert demostraron matemáticamente la incapacidad del Perceptrón para resolver problemas de funciones no

lineales, lo que envió a los anaqueles las investigaciones desarrolladas y desincentivó por completo a la academia.

Sin embargo en 1974 Paul Werbos desarrolló la idea básica del algoritmo de aprendizaje backpropagation (*de propagación hacia atrás*); cuyo significado revivió las redes neuronales artificiales en 1985 cuando John Hopfield publicó "*Computación neuronal de decisiones en problemas de optimización",* posteriormente David Rumelhart y G. Hinton reactivaron las investigaciones y usos del mencionado algoritmo de aprendizaje, generándose a partir de allí gran cantidad de nuevas aplicaciones sobre temas de clasificación, pronóstico y automatización.

Anderson desarrolló un asociador lineal de patrones que posteriormente perfeccionó en el modelo BSB (Brain-State-in-a-Box).Simultáneamente, en Finlandia, Kohonen desarrolló un modelo similar al de Anderson y años más tarde generó un modelo topográfico con aprendizaje auto-organizado en el que las neuronas se distribuyen según el tipo de entrada al que responden. Este modelo topográfico, se conoce como mapa auto-organizado de Kohonen, es una de las redes neuronales más ampliamente utilizadas en la actualidad.

Hopfield publicó en 1982 un importante artículo en la Academia Nacional de las Ciencias (Holpfield, 1982) donde presentó una estrecha relación entre los sistemas físicos y las redes neuronales, su concepto clave es considerar la fase de ajuste de las conexiones como una búsqueda de valores mínimos en unos paisajes de energía. Así, las redes utilizadas por Hopfield poseen una arquitectura monocapa cuyas conexiones son modificadas a partir de un algoritmo de aprendizaje basado en la regla de Hebb. Las redes de Hopfield han sido empleadas como memorias auto-asociativas, principalmente para el reconocimiento de patrones. Posteriormente Hinton y Sejnowski desarrollaron el concepto mediante un sistema denominado

"máquina de Boltzmann". El algoritmo para la modificación de conexiones del sistema de múltiples estratos de Hinton y Sejnowski fue una de las aportaciones más importantes de la primera fase de la reemergencia de estos modelos.

La mayor justificación teórica para el empleo de las RNA: su propiedad de "aproximadores universales" de funciones, fue definida en 1989 por Hornik, Stinchcombe y White, para el siguiente año comprobaron también la aproximación de sus derivadas (Hornik, Stinchcombe, & White, 1990), siendo así útiles para problemas donde el proceso generador de datos sea desconocido y/o no posea relaciones lineales.

Además de esto se han adelantado avances con respecto a la superación del concepto de caja negra que supone este tipo de modelos, como el análisis de sensibilidad de los pesos, o del error así como el método numérico, adelantos realizados durante la década de los 90s (Garson, 1991), (Nath, Rajagopalan, & Ryker, 1997), (Zurada, Malinowski, & Cloete, 1994), una revisión más reciente de la propiedad de aproximación universal de las redes se recoge en Scarselli y Chung (Scarselli & Chung Tsoi, 1998).

En la actualidad existen un sinnúmero de grupos de investigación sobre inteligencia artificial y RNA, todos con enfoques y motivaciones de gran diversidad, uno de los mayores grupos de investigación de los últimos años ha sido el grupo PDP (Parallel Distributed Processing) formado por Rumelhart, McClelland y Hinton. Rumelhart de la Universidad de Stanford es uno de los principales precursores de la red más utilizada, la famosa red neuronal Backpropagation.

Por otra parte destacan las investigaciones de los grupos de California Institute of Technology, Massachussets Institute of Technology, University of California Berkeley y University of California San Diego. Además de los

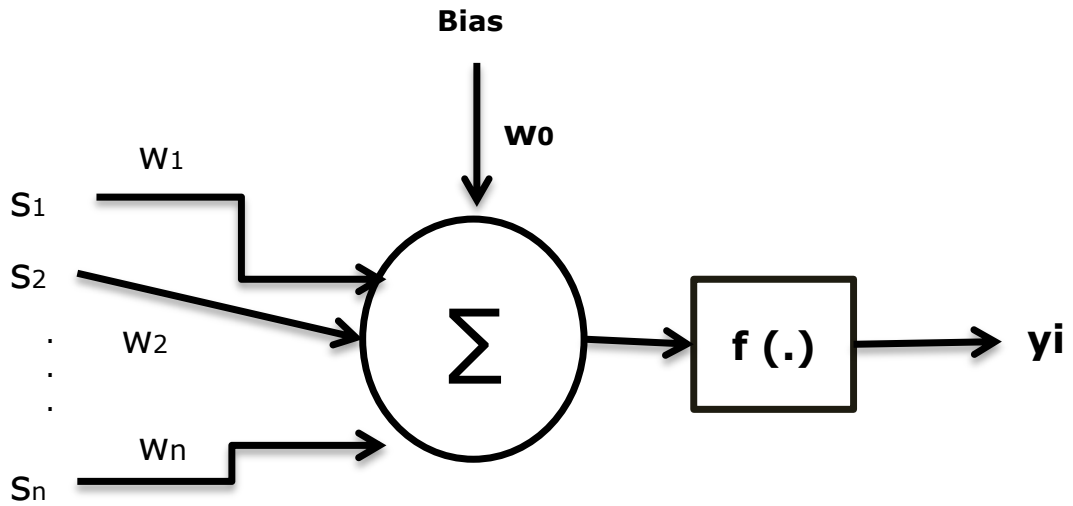
esfuerzos económicos y de investigación que realizan empresas privadas en Estados Unidos, Japón y la Unión Europea.

II.4. CARACTERIZACIÓN

Las RNA son modelos que pretenden emular el comportamiento del cerebro, para lo cual realizan una simplificación usando los elementos básicos del sistema y dejando a manos del diseñador la elección de características y estructura que se correspondan con el problema a enfrentar. Es por ellos de gran importancia entender las formas como se pueden organizar y conectar las redes, estás se distribuyen en capas o niveles, teniendo elementos de entrada, procesamiento o capas ocultas y elementos de salida; como muestra la

Ilustración 2, cada nivel posee determinado número de unidades procesadoras (neuronas), que se estructuran en capas y estas a su vez en red.

Ilustración 2. Neurona artificial



Fuente: Elaboración propia

Las neuronas artificiales se componen de:

a) Sinapsis: Representa las conexiones, está determinada por el valor de las entradas de la neurona ponderadas por un peso w_{ij} que conecta la entrada s_j de la neurona i, incluye también un peso independiente w_{i0} mejor conocido como bias, donde se considera una entrada constante para la neurona i. La sinapsis determinan el comportamiento de la neurona, estas pueden ser excitadoras (presentan un signo positivo), o inhibidoras (conexiones negativas).

b) Sumador: Simula el cuerpo de la neurona y obtiene el nivel de excitación, mediante las funciones de entrada que relacionan las entradas con los pesos de las respectivas sinapsis.
Las funciones de entrada más comunes son:

- Sumatoria: suma de todos los valores de entrada a la neurona, multiplicados por sus correspondientes pesos.

Ecuación 1. Función de entrada Sumatoria

$$\sum_{j=1}^{n}(in_j * w_n)$$

- Multiplicatoria: Es el producto de todos los valores de entrada a la neurona, multiplicados por sus correspondientes pesos.

Ecuación 2. Función de entrada multilicatoria

$$\prod_{j=1}^{n}(in_j * w_n)$$

- Máximo: Solo considera el valor de entrada más fuerte, previamente multiplicado por su peso correspondiente.

Ecuación 3. Función de entrada Máximo

$$Máx_{j=1}(in_j * w_n)$$

c) Función de activación: Genera la salida si se alcanza el nivel de excitación y restringe el nivel de salida, la salida de la neurona está dada por la expresión:

Ecuación 4. Función de activación

$$y_i = \varphi\left(\sum_{j=1}^{n} \mathrm{w_{ij}s_j} + \mathrm{w_{i0}}\right)$$

El argumento de la función de activación es la combinación lineal de las entradas de la neurona. Si se considera al conjunto de entradas y los pesos de la neurona i como un vector de dimensión (n + 1) [1]; la expresión anterior puede escribirse de la siguiente forma:

Ecuación 5. Función de activación matricial

$$y_i = \varphi\left[W_i^T s\right]$$

Donde $S = [S_1; S_2; \dots ; S_n]^T$; $w_i = [w_{i0}; w_{i1}; \dots ; w_{in}]^T$ y φ representa la función de activación, las principales funciones de activación son:

Ecuación 6. Función de activación lineal

$$y(i) = \begin{cases} Si(entrada_i - \theta_i) \ \leq -\frac{1}{a}, & -1 \\ Si\ -\frac{1}{a} < (entrada_i - \theta_i) < \frac{1}{a}\ , & a * (entrada_i - \theta_i) \\ Si\ \frac{1}{a} \geq (entrada_i - \theta_i), & 1 \end{cases}$$

[1] n es el número de entradas a la neurona i

$\boldsymbol{\theta_i}$ Representa el umbral del número positivo, cuyo valor está definido por la pendiente de la función.

Ecuación 7. Función de activación sigmoidea

$$y(i) = \frac{1}{1 + e^{-a*(entrada_i - \theta_i)}}$$

Ecuación 8. Función de activación tangente hiperbólica

$$y(i) = \frac{e^{a*(entrada_i - \theta_i)} - e^{-a*(entrada_i - \theta_i)}}{e^{a*(entrada_i - \theta_i)} + e^{-a*(entrada_i - \theta_i)}}$$

d) Salida: La salida de la neurona puede ser de dos tipos:

Identidad: Dónde la salida es el mismo valor de activación de la neurona

Ecuación 9. Función de salida identidad

$$X(i) = y(i)$$

Binaria: Es una función de respuesta binaria, 0 o 1, utilizada por lo general para problemas de clasificación

Ecuación 10. Función de salida binaria

$$X(i) = \begin{cases} Si\ y(i) \geq \varepsilon_i, & 1 \\ caso\ contrario, & 0 \end{cases}$$

II.5. FUNCIONAMIENTO

Las RNA están basadas en elementos de cálculo relativamente simples, que son capaces de aprender de su ambiente y modificar su forma de interactuar con él. La manera como las neuronas se organizan en la red se llama Topología y las redes se clasifican según su arquitectura.

Si ninguna de las salidas de las neuronas es entrada de un elemento de la misma capa o de capas anteriores, es una red de propagación hacia adelante, cuando las conexiones se dan hacia atrás, adelante o inclusive con ellas mismas, la red se denomina de retro-propagación (backpropagation).

Las redes neuronales artificiales deben ser entrenadas, empleando un grupo de datos de ejemplo con entrada y salida que permitan a la red mediante una regla de aprendizaje ajustar los pesos acorde con las reglas de activación empleadas. En general la potencia de procesamiento de una RNA se mide por el número de interconexiones actualizadas por segundo durante el proceso de entrenamiento o aprendizaje.

La arquitectura de la red se origina en la organización del sistema en procesamiento paralelo y las interconexiones de las neuronas, aunque estas unidades estén diseñadas para ajuste de entradas mediante funciones de activación y salidas como sumas de múltiples entradas ajustadas por los diferentes pesos.

La RNA "aprende" las reglas de procesamiento de los datos mediante reglas de aprendizaje que le permiten ajustar el peso de las conexiones en respuesta la información empleada como ejemplo. Una vez hecho el

aprendizaje, se contrasta para su validación y realimentación del proceso de creación.

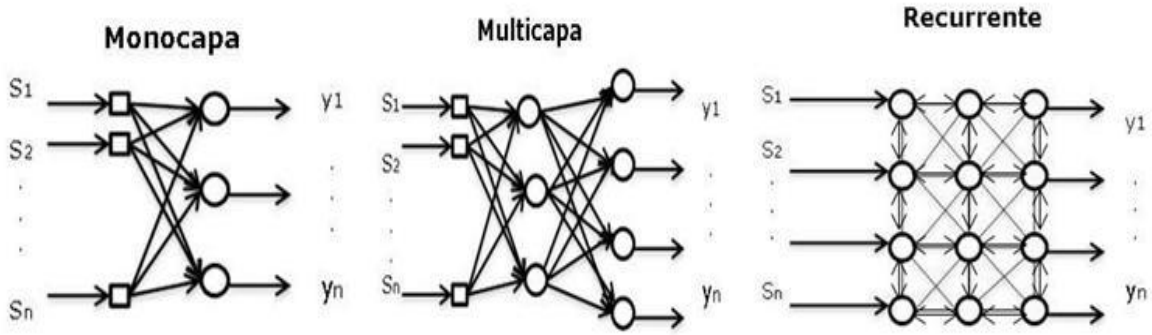
Hay cuatro etapas para la generación de la red, la fase diseño, la fase de entrenamiento, la fase de prueba y la fase de ajustes, a saber:

FASE DE DISEÑO:

En la primera etapa se propone una arquitectura con detreminado número de capas y neuronas, así como se eligen las funciones de activación, la topología de la red y la estructura de conexión.

> **Principales topologías:** La topología de la red define la forma de organización estructural, esta determina en gran medida la capacidad de procesamiento del sistema, por lo general las redes más complejas y más grandes ofrecen mejores prestaciones en el cálculo computacional aunque demandan mayor capacidad de procesamiento que las redes simples. Es importante destacar que la literatura recomienda hacer uso del principio de parsimonia [2], tratando de evaluar el aporte marginal de aumentar la complejidad.

Ilustración 3. Clases de RNA según topoligías



Fuente: Elaboración propia

[2] La Navaja de Ockahama define el principio de Parsimonia como: "*en igualdad de condiciones, la explicación más sencilla suele ser la correcta*"

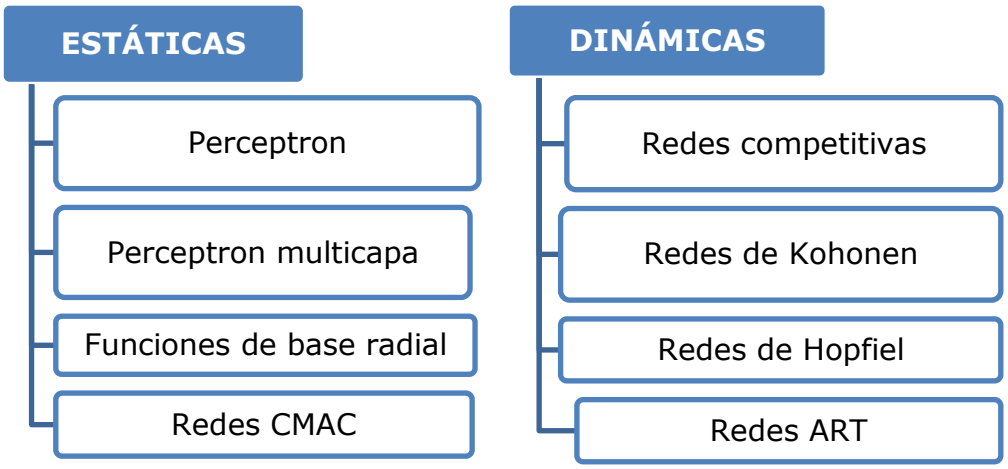
a) Redes monocapa: Estas son la redes más simples presentan un solo nivel con conexión hacia adelante

b) Redes multicapa: se forman con un grupo de capas simples en cascada, la salida de una capa es la entrada de la siguiente capa

c) Redes recurrentes: Están constituidas por múltiples capas donde las neuronas presentan conexiones en todos los sentidos, hacia adelante, hacia atrás con no neuronas de su mismo nivel

Conexión: Un elemento esencial en la estructura de una RNA es la manera en que la información se transmite en la red, esta se clasifica en dos grupos: redes con conexiones hacia adelante (estáticas) y redes con conexiones hacia atrás ó recurrentes (dinámicas).

Ilustración 4. Clasificación de la red según conexión



Fuente: Elaboración propia

Función de activación: Es la encargada de calcular el estado de activación de una neurona, como se comentó en el apartado anterior, las más habituales son la función lineal Ecuación 6, la sigmoideal Ecuación 7 y la tangente hiperbólica Ecuación 8.

Una propiedad de las funciones sigmoidal y tangente hiperbólica es que su derivada existe y es una función continua. Esto es necesario ya que en el algoritmo de aprendizaje el cálculo del gradiente local para cada neurona del perceptrón multicapa requiere de la determinación de la derivada de la función de activación asociada con la neurona en cuestión. Para que esa derivada exista es necesario que la función sea diferenciable. Las funciones de activación comúnmente utilizadas en redes neuronales son las funciones sigmoidal y gaussiana.

Regla de aprendizaje: Se debe fijar una regla de aprendizaje que permita el ajuste de los pesos de las conexiones, hasta obtener una estructura favorable, esto emula el conocimiento que en el cerebro se halla en la sinapsis, este tema se explicará en el siguiente apartado.

FASE DE ENTRENAMIENTO:

El objetivo de esta etapa es hacer que la información de referencia, ordenada como variables en vectores, sea procesada bajo reglas de aprendizaje y ajuste iterativo de los pesos en las sinapsis, convergiendo gradualmente a los vectores de salida del conjunto de datos. El entrenamiento se puede realizar hacia adelante, desde las variables de entrada propagadas hacia la salida, comparándola con el valor deseado, o hacia atrás, donde a partir del error se configuran los pesos sinápticos mediante retro-propagación del error desde la capa de salida hacia los valores de entrada.

Los procedimientos de ajuste son regidos por los algoritmos de entrenamiento, clasificados en:

a) **Entrenamiento supervisado**: Para calcular el error y utilizarlo para retroalimentar la red y ajustar los pesos de acuerdo con el algoritmo, se realiza una comparación de cada vector de entrada con su respectiva estimación de salida en el entrenamiento, teniendo como objetivo la miminzación del error. Dentro de este hay tres métodos de aprendizaje.

   - **Aprendizaje por corrección de error:** Ajusta los pesos de las conexiones en función de la diferencia entre los valores deseados y el calculado en la salida de la red.

   - **Aprendizaje por refuerzo:** No se indica la salida exacta durante el entrenamiento, lo que se hace es que el supervisor genera una señal de refuerzo cuando la salida se ajusta al valor deseado y en función de un ajuste por probabilidades se re calibran los pesos, suele ser el más lento de los aprendizajes

   - **Aprendizaje estocástico:** Se realizan cambios aleatorios en los pesos de las conexiones y se evalúa el error en la estimación generando distribuciones de probabilidad que se usan para la elección de los pesos óptimos.

   En general este tipo de entrenamientos son los más utilizados, pues arrojan buenos resultados en el proceso de entrenamiento de los diferentes modelos. Sin embargo, han sido fuertemente criticados por no poseer mucha lógica desde el punto de vista biológico y en el entrenamiento de algunos procesos específicos, por ello se han

desarrollado métodos de entrenamiento no supervisados que se explican a continuación.

b) Entrenamiento no supervisado: Son modelos de aprendizaje basados en la lógica, no se corresponden con un vector de salidas y se busca generar pesos que generen resultados consistentes, mediante propiedades estadísticas del conjunto de vectores de entrenamiento agrupa en clases a los vectores similares. Existen dos tipos conocidos de entrenamiento no supervisado:

- ***Aprendizaje hebbiano:*** Cosiste en el ajuste de los de pesos de acuerdo con la correlación (+1 y -1) de las salidas con las neuronas interconectadas, si dos acciones son activas (positivas) la conexión se refuerza, mientras que si una es activa y otra negativa la conexión se debilita.

- ***Aprendizaje competitivo o cooperativo:*** Es una "competencia" presente en todas las capas de la red, cuando cada neurona tiene conexiones recurrentes de autoexcitación y conexiones de inhibición, de esta manera datos similares formaran parte de la misma categoría y activarán la misma neurona de salida.

  Las categorías son creadas por la misma red, el aprendizaje solo afecta a las neuronas activas, redistribuyendo el peso total entre las conexiones que llegan a la neurona "ganadora" y repartiendo el peso entre todas las conexiones, así la neurona *a* se modifica solo si es excitada por la neurona precedente *b.*

c) Entrenamiento reforzado: Sucede cuando la información es generada por la interacción de un agente con el ambiente en cada tiempo T, bajo determinada dinámica, de esta manera el objetivo es estipular las acciones que minimizan el costo del aprendizaje a largo plazo. Estos procesos se suelen simular mediante procesos de Markov

donde los estados son S_i, las interacciones son A_i las distribuciones de costo son $P(C_i|S_i)$, y las acciones se definen como las distribuciones condicionales sobre las interacciones observadas, este tipo de soluciones se emplea en problemas de control y aprendizaje para inteligencia artificial.

Pero la materialización del entrenamiento se logra gracias a la implementación de algoritmos de aprendizaje, son reglas que permiten realizar la reubicación de los pesos y optimizar la salida.

A continuación se hace una breve presentación de los algoritmos comúnmente empleados. Su utilización está sujeta al contexto general del modelo a plantear, queda a criterio del diseñador la elección de aquel que mejor responsda a sus necesidades, teniendo en cuenta:

- Eficacia
- Eficiencia computacional
- Parsimonia
- Ajuste al problema y sus variables
- Facilidad en el análisis a realizar

ALGORTIMOS DE APRENDIZAJE

Retro-propagación (Backpropagation): Por lo general se emplean algoritmos que optimizan conexiones a partir de la minimización del error entre la salida generada y el valor deseado. Los más populares son los algoritmos por descenso del gradiente, basados en la minimización o maximización de una función de error, por lo general se suele minimizar una función del error monótona creciente (la más común es el Error cuadrático medio -mse-) ya que supone una distribución normal de los errores y responde bien a los problemas de modelización.

Ecuación 11. Error cuadrático medio

$$J = \frac{1}{2M}\sum_{i=1}^{M}\sum_{j=1}^{\mathrm{N}} \mathrm{e_j^2}\ (\mathrm{i})$$ (Bishop, 1995)

N es el número de neuronas, M el número de variables, por lo general se emplea en problemas de modelización.

Existen otras alternativas presentadas por Bishop (Bishop, 1995), como la función de coste entrópica que supone una distribución de errores de tipo binomial, útil en el modelado de procesos de clasificación o variables de respuesta dicotómica Ecuación 12 o la norma de Minkowski de la Ecuación 13 que permite minimizar el error cuadrático al inicio del entrenamiento (R=2) y el valor absoluto del error (R=1) al final del entrenamiento.

Ecuación 12. Función de coste Entrópica

$$J = \frac{1}{2M}\sum_{i=1}^{M}\sum_{j=1}^{\mathrm{N}}\left[\left(1+\mathrm{d_j(i)}\right)\ln\left[\frac{1+\mathrm{d_j(i)}}{1+\mathrm{o_j(i)}}\right]+\left(1-\mathrm{d_j(i)}\right)\ln\left[\frac{1+\mathrm{d_j(i)}}{1+\mathrm{o_j(i)}}\right]\right]$$

Ecuación 13. Norma de Minkowski

$$J = \frac{1}{R\,N}\sum_{i=1}^{L}\sum_{j=1}^{\mathrm{N}}\left|\mathrm{d_j(i)}-\mathrm{o_j(i)}\right|^{\mathrm{R}}$$

Momentos, Regla Delta Generalizada: Un gran inconveniente del método Backpropagation es que su objetivo es minimizar el error, pero por lo general la función suele tener varios mínimos locales, y se puede caer en uno de estos puntos no deseados, una alternativa para sortear esta dificultad es la Regla Delta Generalizada (Ecuación 14), que propone que los cambios de los pesos sinápticos dependan del gradiente medio de los puntos de un entorno, y no de un mínimo, teniendo en cuenta el gradiente actual y el de la iteración anterior Hinton y Williams (1986) para reducir las fluctuaciones de los gradientes próximos, ya que la variación errática que

en ocasiones presenta el gradiente genera grandes costos computacionales e ineficiencias en los algoritmos.

Ecuación 14. Regla delta generalizada

$$\Delta w_j(k) = \alpha \Delta w_j(k-1) - \eta \frac{\partial E}{\partial w_j(k)} = \Delta w_j(k-1) - \eta \delta^2(\mathrm{k}) s_j(k)$$

Esta ecuación representa la variación del componente i para la iteración k, donde K representa la iteración actual, α es constante de momentos ($0 \leq \alpha < 1$), controla el grado de modificación de los pesos teniendo en cuenta la modificación en la etapa anterior y η es la tasa de aprendizaje, esta última debe ser fijada con cautela, ya que una tasa demasiado pequeña se garantiza que la función de error disminuya cuando se va en sentido contrario a la dirección del gradiente, pero la convergencia del algoritmo será mucho lenta debido a que se dan pequeños paso, si la tasa de aprendizaje es demasiado grande el tiempo de computo será inferior pero el algoritmo oscilará aumentado y disminuyendo la función de error y no alcanzará un mínimo.

Levenberg-Maquardt: Esta técnica se basa en la localización del mínimo de una función multivariada que es la suma de los cuadrados de una función real no lineal, suele ser una función rápida, robusta y que proporciona convergencia, sin embargo requiere una mayor capacidad computacional respecto a las anteriores propuestas, como la presentada a continuación:

Ecuación 15. Algoritmo de Levenberg-Maquardt

$$E(w) = \frac{1}{2} \sum_{i=1}^{I} [z_i - y_i(w)]^2 = \frac{1}{2} e(w)^t e(w)$$

Una de las versiones más comunes de esta ecuación es la del Error cuadrático medio, que es la que suele venir programada en los diferentes softwares como la Ecuación 11, donde e_j^2 representa el error de entrenamiento de la salida j que se define en la Ecuación 16, donde $d_{p,m}$ es el vector de salida deseado y o_{pm} el vector de salida real.

Ecuación 16. Error de entrenamiento

$$e_{p,m}^2 = d_{p,m} - o_{pm}$$

Se debe destacar que la utilización de este algoritmo supone también la inclusión de una tasa de aprendizaje que enfrenta las limitaciones ya mencionadas.

FASE DE PRUEBA:

Por lo general los datos que se poseen se dividen de manera aleatoria en dos grupos, esto para poder dedicar una gran proporción a la etapa de entrenamiento y el grupo restante a probar la red entrenada, en esta etapa se usa la red estimada para un grupo de datos y se contrasta la salida de la red con el valor real que toma la variable, el error que se obtiene sirve como referente para validar la capacidad de la red ante datos nuevos, el ajuste de los parámetros y si existe o no sobre-aprendizaje (problema común donde la red no aprender sino que memoriza los casos presentes en la información de entrenamiento). Sin embargo, algunos programas incorporan un tercer grupo de control, como se explicará en la metodología empelada, para poder pruebas de sensibilidad de la respuesta ante cambios en los valores procesados.

Una vez programada la estructura de la red en el software que se empleará para su computación se deben definir los diferentes criterios para la detención del entrenamiento como son el tiempo, un error mínimo, un

mínimo en la velocidad de aprendizaje, determinado número de iteraciones, entre otros.

FASE DE AJUSTES:

En la actualidad no se ha desarrollado ninguna metodología o mecanismo de diseño óptimo de la red, es por eso de vital importancia en el proceso de diseño realizar una tarea de ajuste iterativo, acorde con los resultados, variando la topología, algunas partes del diseño, entre otras y contrastando hasta llegar a aquella que minimice el error de manera óptima.

Una aproximación a la mejora de la red es el análisis de Poda, que consiste en ir reduciendo el número de variables y contrastando su efecto, así como variar (aumentar o disminuir) el número de capas y neuronas de manera organizada y comparando los resultados de los cambios propuestos.

Los procedimientos más comunes para la realizar el proceso de ajuste por Poda son:

- Reducir la red en número de neuronas o capas y quedarse con aquella que minimice el error cuadrático medio en la validación o quitar aquellas cuyo peso sináptico convertido a cero no afecte significativamente dicho error.

- Eliminar las variables cuyo descarte no genere cambios significativos en caso de ser descartadas.

- Una manera más sofisticada de realizar el proceso de Poda mediante el método OBD (*optimal brain damage*) (Hashem, 1997), propuesto por (Le Cun, Denker, & Solla, 1990), consiste en identificar los pesos que deben ser eliminados mediante las derivadas de segundo orden de la función de error contenida en la matriz Hessiana, descartando aquellos cuya variación genere el menor error.

Pese a los años de investigación y los múltiples esfuerzos, muchas personas consideran que las RNA son un caja negra, por su configuración dentro de las capas ocultas ante esto, se han desarrollado diversos análisis en paralelo que permiten evaluar los resultados de la red y el comportamiento de las variables dentro de esta. A continuación se presentan los análisis más empleados en la actualidad.

II.6. ABRIENDO LA CAJA NEGRA

Análisis basado en la magnitud de los pesos

Este análisis tuvo origen en la consideración del valor absoluto de los pesos, dónde los más altos suponían mayor importancia, sin embargo con el tiempo se demostró que esta afirmación no era correcta, ya que los pesos están sesgados por las magnitudes de las variables y la configuración topológica de la red.

Ante esto Garson propuso una expresión más robusta, presentada en la Ecuación 17, que considera la influencia relativa de cada variable de entrada sobre cada una de las salidas de la red.

Ecuación 17. Magnitud de pesos

$$Q_{iy} = \frac{\sum_{j=1}^{L} \frac{W_{ij}}{\sum_{r=1}^{N} W_{ij}} v_{jy}}{\sum_{i=1}^{N} \sum_{j=1}^{L} \frac{W_{ij}}{\sum_{r=1}^{N} W_{ij}} v_{jy}}$$

Donde i es la i- ésima neurona de la capa de entrada, j es la j-ésima neurona de la capa oculta, N el número de variables de entrada, L es el

número de neuronas de la capa oculta, W_{ij} es el peso que conecta la capa de entrada de la variable i con la neurona j, v_{jy} son los pesos de conexión entre la capa oculta y la capa de salida. Los pesos se deben usar en valor absoluto, el valor umbral (bias) de las neuronas ocultas y de salida no se tienen en cuenta, se asume que su inclusión es despreciable a efectos generales (Garson, 1991).

El índice Q_{iy} representa el porcentaje de influencia de la variable *i* sobre la salida *y*, en relación a las demás variables de entrada, por lo que la suma de los Q_{iy} debe ser igual a 100%.

Análisis de sensibilidad basado en el error

Este consiste en realizar un análisis del efecto producido sobre el error de la red neuronal ante cambios en las variables de entrada, la función a usar es la expresada en la Ecuación 18.

Ecuación 18. Raíz cuadrada del erro cuadrático medio

$$\boldsymbol{RSME = \sqrt{\frac{\sum_{p=1}^{P}\sum_{k=1}^{M}(d_{p,y} - o_{p,y})^2}{P * M}}}$$

Donde $d_{p,k}$ es la salida objetivo para el patrón p en la neurona de salida *y.* La implementación de este análisis se realiza a partir de pequeños incrementos de las variables de entrada a lo largo de todo su rango, ceteris paribus las demás variables (Frost & Karri, 1999), se calcula el error en cada RNA estimada y se repite el procedimiento para cada variable, de esta manera aquella cuyos errores sean mayores será la variable con mayor importancia para las variables de salida y aquella de menor error será la que menos aporte realiza.

Existe una limitación a la hora de hacer uso de este método y es que la generación de los incrementos supone el uso de variables continuas, ya que en el caso de variables discretas no se pueden generar los pequeños incrementos requeridos, reduciendo su empleabilidad a problemas que contengan entradas continuas.

Análisis de sensibilidad basado en la salida

Este consiste en estudiar el efecto que causa en la salida el cambio de una variable de entrada. Sobre la red entrenada se fijan los promedios de las variables y se empieza a incluir ruido en estas, registrando los cambios en la salida de la red y aplicando un índice de resumen de estos cambios.

Una apliación de este método se puede realizar sumando el R SME de la estimación ante una gran variedad de cambios realizados en una variable, ceteris paribus y comparando dichas sumatorias, se entenderá que aquella cuya sumatoria de R MSE sea mayor es más sensible para el modelo.

Este análisis tiene la ventaja de ser práctico e intuitivo pero a la vez carece de robustez y es sensible al criterio de quién lo esté empleando.

Matriz de sensibilidad Jacobiana

La matriz Jacobiana *S* permite de manera analítica determinar un indicador de sensibilidad de las salidas ante el cambio producido por las variables de entrada, en la matriz $S_{(IxK)}$, I representa el número de entradas y K el número de salidas, el elemento S_{ik} representa la sensibilidad de la salida k con relación a la variable *i.* Estos elementos se obtienen realizando la derivada parcial de una salida y_k con respecto a una entrada x_i, como muestra la Ecuación 19.

Ecuación 19. Cálculo de los elementos de la matriz Jacobiana

$$S_{ik}\frac{\partial Y_k}{\partial X_i}$$

El nivel de importancia de la variable estará determinado por el valor absoluto de su respectivo valor en la matriz Jacobiana, el signo se relaciona con la relación que tiene la entrada con la salida.

Sin embargo en casos donde el valor estimado d_k se aproxima al observado o_k entonces el término derivativo $f'(net_k)$ es próximo a cero para las funciones sigmoidales que suelen ser comúnmente empleadas. Por tanto, el valor de la derivada parcial queda anulado ante esto han surgido varias soluciones, una alternativa es suprimir el término derivativo $f'(net_k)$ asumiendo que no afecta a la comparación entre las sensibilidades de las diferentes variables respecto a la salida, la cual fue propuesta por (Lisboa, 1994); otra alternativa es emplear funciones lineales en la capa de salida, de esta manera la derivada parcial seria igual a 1.

Esta técnica depende de la información aprendida, de la arquitectura y la topología de la red, así que, la sensibilidad que evalúa depende de todo el proceso que implica la generación de la red. Entonces se puede definir la sensibilidad a partir de la media y desviación de S_{ik} y para cada fila de la matriz.

Sin embargo diferentes estudios empíricos realizados con esta técnica encuentran limitaciones por su origen en propiedades estáticas tal como lo mostró (Gedeon, 1997), al comparar los resultados de análisis con la matriz Jacobiana y los análisis de sensibilidad ya mencionados.

Método de sensibilidad numérico

Este método calcula las pendientes que se forman entre entradas y salidas. Para analizar el efecto de una variable de entrada x_i sobre una variable de salida y_k, se deben ordenar de manera ascendente los p patrones a partir de los valores de la variable de entrada x_i, para ello se genera un número *G* de grupos de más o menos igual tamaño, sujeto al número de patrones disponible y la complejidad de la función que se establece entre la entrada y la salida. A cada grupo se le valora los x_i y y_k promedio, con esto se puede calcular el índice NSA, tal como lo explica la Ecuación 20.

Ecuación 20. Método de sensibilidad numérico

$$NSA_{ik}(g_r) = \frac{E\big(y_k(g_{r+1})\big) - E(y_k(g_r))}{E\big(x_k(g_{r+1})\big) - E(x_k(g_r))}$$

Luego de calcular los G-1 índices NSA, se obtiene el valor de la pendiente entre la variable de entrada *i* y la variable de salida *k* como la esperanza del índice, este valor representa el efecto medio que tiene un incremento de x_i sobre y_i, si la variable de entrada es discreta se puede calcular como el coeficiente entre los respectivos rangos, y representa el efecto promedio provocado por el cambio del valor mínimo al valor máximo en la variable x_i. A mayor valor absoluto de $E(NSA_{ik}(g_r))$, más importante es la influencia que tiene la variable en la salida, así como en la matriz Jacobina el signo indica si el cambio observado en y_k va en el mismo sentido o no que el cambio provocado en $x_{i.}$

En general este método supera las principales limitaciones de los métodos anteriormente mencionados, pero no cuenta con tantas comprobaciones empíricas como aquellos.

II.7. VENTAJAS

Dentro de las múltiples ventajas a destacar de los modelos RNA cabe mencionar el potencial computacional gracias a su estructura de cálculo paralelo que le permite la resolución de problemas que necesitarían gran cantidad de tiempo en ordenadores "clásicos", además de ello resaltan otras propiedades a mencionar:

- Adaptabilidad: Gracias al aprendizaje, las neuronas entrenadas a partir de datos de entrada, tienen la capacidad de modificar los parámetros de acuerdo a los cambios presentes en las entradas del sistema, sin embargo esta no se puede considerar excesivamente grande ya que esto conlleva a inestabilidades en el sistema, eso es conocido como el dilema Plasticidad-Estabilidad.

- Auto organización: Gracias a la capacidad de aprendizaje adaptativo son capaces de organizar la información que reciben durante el aprendizaje, creando su propia organización de la información obteniendo así la facultad de responder apropiadamente cuando se les presentan datos diferentes a los empleados en el entrenamiento.

- Sistemas distribuidos no lineales: Las neuronas son elementos no lineales por lo que sus conexiones también son no lineales, esto permite abordar sistemas caóticos, lineales o no lineales, lo que permite en cierto sentido abarcar los modelos econométricos habituales como Regresiones Lineales, Modelos Lineales Generalizados, Regresiones Logísticas, entre otras.

- Solución de problemas no lineales: Las redes neuronales son capaces de relacionar dos conjuntos de datos mediante relaciones complejas.

- Operación en tiempo real: Gracias a su funcionamiento en paralelo, se emplean en procesos realizados en tiempo real. Aunque este tipo de aplicaciones requiere una gran capacidad de procesamiento, se

diseñan y fabrican maquinas con hardware especial para estos procesos.

- Tolerancia a fallos: Al ser sistemas de procesamiento paralelo, permiten el fallo de algunas variables de entrada sin alterar significativamente la respuesta del sistema.

- Fácil inserción dentro de la tecnología existente: En la actualidad múltiples programas y hardware de uso privado y libre incluyen herramientas de ayuda o programación para el diseño, entrenamiento, validación e implementación de estos sistemas en diversas aplicaciones.

II.8. APLICACIONES

Las principales aplicaciones abarcan simples clasificadores, modelos de pronostico, modelos de control, optimizadores, procesamiento de datos a información entre otras, la mayoría se han realizado con miras al desarrollo del concepto de inteligencia artificial (IA) y aplicaciones ingenieriles, donde destacan: conversión de texto a voz, reconocimiento de imágenes y caracteres, procesamiento de señales, filtros de ruido, simuladores de software, aceleradores de hardware, servoControl, chips de silicio, optimización de procesos electrónicos, entre otros muchos.

Pero gracias a su capacidad como modelo de pronóstico también resaltan aplicaciones en las ciencias naturales, en campos como medicina, química y sicología tales como análisis y predicción de control de enfermedades, pruebas para medicamentos, simulación de experimentos, modelos de pronóstico y diagnóstico de enfermedades sicológicas, etc.

Pero las aplicaciones de interés para este trabajo son las realizadas en el área de finanzas y actuarial, donde destacan modelos de predicción de insolvencia, detección de fraude, modelos de clasificación de riesgo y asegurados, modelos de longevidad, análisis de la demanda, riesgo de suscripción, modelos de Bonus Malus, entre otros. A continuación se hace una mención de algunas publicaciones destacadas en estos temas:

- "Soft Computing Applications in Actuarial Science" (Shapiro, 2000) presenta las técnicas de la lógica difusa, algoritmos genéticos y redes neuronales, esta última como herramienta eficaz para el modelado de problemas de clasificación de la demanda, modelos de estimación de insolvencia y pronostico del comportamiento de activos e inversiones, además destaca el potencial en otras áreas como estudios de mortalidad, morbilidad y pricing.

- "Artificial Neural Networks used in automobile insurance underwriting" (Kitchens F. L., 2005) desarrolla una red neuronal, a partir de trabajos previos del mismo autor, para determinar el riesgo de suscripción de pólizas para coches.

- "An application of unsupervised neural networks in general insurance: the determination of tariff classes" (Pelessoni & Picech, 2002) Realizaron un análisis de conglomerados, explotando las propiedades topológica de mapas auto organizativos de Kohonen para generar categorías de tarifas que permita clasificar tarifas con valores contiguos.

- "Factores de riesgo y cálculo de primas mediante técnicas de aprendizaje" (Bousoño, Heras, & Piedad, 2008) Emplean técnicas de aprendizaje máquina (Algoritmos genéticos, Árboles de decisión y Maquinas de vectores) para solucionar el problema de clasificación de nuevos clientes y su distribución de riesgo subyacente, a partir de datos históricos de entrenamiento, identificando variables de gran aporte a la variabilidad del riesgo.

- El documento de trabajo publicado por (Actuarial Considerations ApSTAT technologies, 2003) realiza una revisión conceptual de las redes neuronales, resalta su potencialidad dentro de las ciencias actuariales, centrándose en una interpretación matemática de cómo las redes neuronales pueden capturar dependencias de alto nivel y establece una conexión con la teoría de credibilidad.

- "Teoría de las redes neuronales para la gerencia de riesgos" (Ricote Gil, 2003) Contiene una revisión de las aplicaciones de redes neuronales a la gerencia de riesgos para las compañías de seguros y entidades financieras para conocer y analizar los elementos del riesgo, los sistemas de jerarquía, las asociaciones de riesgos y la segmentación.

- "An application of neural networks to insurance underwriting" (Kitchens, Booker, & Rebman, 2005) Contiene un estudio comparativo para datos de suscripción de una empresa aseguradora modelados mediante regresión lineal, logística y RNA, comparando sus resultados con matrices de confusión, y la prueba de rangos con signo de Wilcoxon, concluyendo la superioridad del modelo generado con RNA, exaltando el potencial de esta herramienta pero reconociendo la distancia existente entre aplicaciones específicas y una RNA actuarial universal, así como anima a continuar con los desarrollos e investigaciones en pro del aprovechamiento de estas técnicas.

- "Modeling for fraud: Neural Networks Demystified" (Francis, 2002) desarrolla un red neuronal clasificatoria para poder realizar análisis de fraude para una cartera de compañía aseguradora, destacando la capacidad de solucionar problemas de clasificación no lineales y los avances en pro de abrir "la caja negra".

III. SEGURO DE COBERTURA PARA EL CULTIVO DE NARAJA Y VARIABLES A EMPLEAR

El sistema de seguro agrario español es uno de los más reconocidos a nivel mundial, por su historia y organización, ha sido un elemento clave para el desarrollo de los diferentes negocios agropecuarios estimulados en el país y representa hoy uno de los pilares del desarrollo agropecuario nacional.

Dentro de sus principales líneas de negocio sobresale el seguro que da cobertura a los cítricos, en especial al cultivo de naranja, concentrándose el 47% de la contratación en la provincia de Valencia.

Uno de los principales puntos clave a la hora de estructurar coberturas para este producto es el desarrollo de un sistema de evaluación del riesgo y tarificación, que aproxime de manera fiable los diferentes riesgos para las niveles de protección, variedades, zonas geográficas y demás especificaciones de las diferentes parcelas aseguradas. En la actualidad la tarifa se fija por riesgo y zona geográfica, y se ajusta para cada póliza por bonus malus y los correspondientes subsidios que pueda recibir.

Las múltiples coberturas ofrecidas se establecen mediante 4 módulos que a su vez se subdividen en tres franquicias absolutas con un mínimo indemnizable y cubren los riesgos de Helada, Viento, Pedrisco y Daños excepcionales. Las tarifas por zona suponen el manejo y corrección de un gran volumen de tasas que luego son ajustadas según las condiciones especiales de cada póliza. Todo este proceso cuenta con todo el background propio de los largos años de experiencia de este seguro, que data de 1981 y la experticia de Agroseguro.

En este marco, se propone un método alternativo de estimación del daño esperado por póliza mediante el desarrollo de redes neuronales artificiales para los riesgos de Helada y Viento, que son los más significativos en frecuencia y severidad. Partiendo de la información de contratación y siniestros desde 1997 hasta 2013 se podrá evaluar el riesgo esperado y, acorde con las diferentes coberturas generar, la tarifa para cada póliza requerida, de esta manera se ofrece, no solo una alternativa para la estimación del riesgo esperado y el cálculo de provisiones, sino además para realizar tarifas personalizadas mediante un cálculo sencillo, y reducir el volumen computacional actual para el procedimiento de tarificación.

Con tal fin, se plantea un acercamiento a un modelo global, utilizando la base de datos de contratación y siniestro por póliza, para la la Comarca Ribera del Jucar para el término de con mayor contratación a nivel nacional, con un total de 1076, 408 registros para los riesgos de Helada y Viento respectivamente, es de aclarar que la diferencia en registros se debe a diferentes intereses de la demanda, inicios de nuevas coberturas y reestructuración de productos a lo largo del tiempo. Se utilizaran las siguientes variables:

III.1. VARIABLES CUALITATIVAS

Final de garantía:

Variable categórica que expresa el periodo de finalización de la cobertura. Como se puede ver en la Tabla 1, la contratación se ha concentrado en las variedades tempranas, pues los finales de garantía con más registros son los de diciembre y enero, el final diseñado para el redrojo ha tenido muy poca acogida y las coberturas varían acorde al riesgo, menos coberturas de helada en febrero pero más de viento en abril, esto acorde con los periodos de máxima probabilidad de ocurrencia de cada riesgo.

Tabla 1. Frecuencia Final de garantía para Viento y Helada

FINALES DE GARANTIA	FRECUENCIA EN VIENTO	PORCENTAJE EN VIENTO	FRECUENCIA EN HELADA	PORCENTAJE EN HELADA
Final de Redrojo	3	0,70%	10	0,90%
Diciembre	112	27,50%	300	28,00%
Enero	122	29,90%	240	22,40%
Febrero	18	4,40%	153	14,30%
Marzo	23	5,60%	23	2,10%
Abril	102	25,00%	250	23,30%
Mayo	28	6,90%	97	9,00%
TOTAL	**408**		**1.073**	

Fuente: Elaboración propia

Variedad:

Variable categórica, compuesta por variedades cultivadas a los largo de la historia de la contratación que se muestran en la Tabla 2, la variedad con mayor contratación es la Newhall ecológica, y aquellas de poca participación han sido incorporadas en el último plan de coberturas, por ello su bajo nivel de acogida, hay algunas variedades que solo han contratado de manera experimental uno de los dos riesgos, por el mismo motivo. También es posible identificar como la helada tiene una inclusión de mayor número de variedades, debido a la gran sensibilidad de las naranjas a este riesgo, haciendo más atractiva dicha cobertura.

Tabla 2. Frecuencia Final de garantía para Viento y Helada

VARIEDAD	FRECUENCIA EN VIENTO	PORCENTAJE EN VIENTO	FRECUENCIA EN HELADA	PORCENTAJE EN HELADA
SANGUINA ECOLOGICA	1	0,20%	12	1,1%

VARIEDAD	FRECUENCIA EN VIENTO	PORCENTAJE EN VIENTO	FRECUENCIA EN HELADA	PORCENTAJE EN HELADA
SANGUINA	0	0,00%	17	1,6%
SANGUINELI	22	5,40%	22	2,0%
STAR RUBY ECOLÓGICA	0	0,00%	12	1,1%
THOMSON NAVEL TRATADA	0	0,00%	5	0,5%
VERNA	21	5,10%	95	8,8%
CLEMENTINA NULES BIOLOGIC	7	1,70%	51	4,7%
HERNANDINA ECOLÓGICA	1	0,20%	12	1,1%
LANE LATE ECOLÓGICA	86	21,10%	187	17,4%
NADORCOTT ECOLÓGICA	0	0,00%	6	0,6%
NAVEL ECOLÓGICA	1	0,20%	0	0,0%
NAVEL TRATADA	2	0,50%	5	0,5%
NAVELATE ECOLÓGICA	1	0,20%	5	0,5%
NAVELATE SIN TRATAR	0	0,00%	5	0,5%
NAVELINA	10	2,50%	10	0,9%
NEWHALL	1	0,20%	9	0,8%
NEWHALL ECOLÓGICA	215	52,70%	448	41,7%
NOVA (CLEMENV) ECOLÓGICA	1	0,20%	13	1,2%
OROGRANDE	5	1,20%	64	6,0%
OTRAS CLEMENTINAS	0	0,00%	39	3,6%

VARIEDAD	FRECUENCIA EN VIENTO	PORCENTAJE EN VIENTO	FRECUENCIA EN HELADA	PORCENTAJE EN HELADA
H. C.				
REDROJO DEL MESERO	20	4,90%	30	2,8%
OTRAS VARIEDADES	14	3,40%	28	2,6%
TOTAL	**408**		1075	

Fuente: Elaboración propia

VARIABLES CUANTITATIVAS:

Superficie asegurada:

Esta variable representa las hectáreas aseguradas por póliza. Como permite ver la Tabla 3, los valores son mucho más dispersos en el caso de helada, su media es de 0,78 has, estando mucho más concentrada en a la izquierda de la distribución que viento, pero con mayor cantidad de valores atípicos.

Tabla 3. Estadísticos descriptivos de Superficie asegurada

SUPERFICIE (Hectáreas)		VIENTO	HELADA
N de datos		408	1.075
Valores perdidos		0	0
Media		1,738	0,781
Moda		0,250	0,470
Desviación estándar		2,820	2,980
Varianza		7,955	8,881
Rango		34,981	59,94
Mínimo		0,021	0,06
Máximo		35,000	60
Percentiles	**25**	0,370	0,34

SUPERFICIE (Hectáreas)		VIENTO	HELADA
	50	0,795	0,47
	75	1,89	0,58

Fuente: Elaboración propia

Valor de producción:

Representa el valor de producción asegurado en euros, sus estadísticos descriptivos se encuentran en la Tabla 4. En el caso de helada es una variable un poco más dispersa, aunque en ambos casos es de cola larga y asimétrica a la derecha, concentrando sus valores en los dos primeros cuartiles y con grandes datos atípicos. El comportamiento de viento es similar pero con un rango y desviación inferiores.

Tabla 4. Estadísticos descriptivos de Valor de producción

VALOR DE PRODUCCIÓN (€)		VIENTO	HELADA
N de datos		408	1075
Valores perdidos		0	0
Media		12.8990,997	17.377,97
Moda		6000	13.000
Desviación estándar		25.811,57	26.645
Varianza		666.200.000	709.900.000
Rango		249.982,00	259.999,84
Mínimo		18	18
Máximo		25.0000	260.000,02
Percentiles	25	2.000	3.744,00
	50	5.000	9.000,00
	75	11.700	19.500,00

Fuente: Elaboración propia

Precio:

Representa el precio unitario por kilo de la naranja, es regulado cada año por el Ministerio de Agricultura y se determina por variedad y destino comercial (calidad o industria). La Tabla 5 muestra comportamientos similares para las bases de datos de Viento y Helada, el precio está concentrado, en general, no presenta valores atípicos y refleja el fuerte control que ha ejercido el gobierno en estos años sobre la precio, es importante aclarar que acorde con este se valoran los daños y a su vez las indemnizaciones pagadas.

Tabla 5. Estadísticos descriptivos de Valor de producción

PRECIO (€)		**VIENTO**	**HELADA**
N de datos		408	1075
Valores perdidos		0	0
Media		0,1465	0,21
Moda		0,30	0,18
Desviación estándar		0,10279	0,11
Varianza		0,011	0,01
Rango		0,42	0,7
Mínimo		0,01	0,01
Máximo		0,42	0,7
Percentiles	**25**	0,0547	0,15
	50	0,126	0,18
	75	0,2273	0,27

Fuente: Elaboración propia

Daños por Viento:

Esta es una de las dos variables a explicar, representa los daños estimados en euros por póliza a causa del viento.

Dentro de las condiciones generales del seguro para el cultivo de naranaja se define el viento como: *Movimiento de aire, que por su intensidad y persistencia, ocasione por acción mecánica pérdidas directas en los bienes asegurados.*

Además, *para qu exista garantía sobre la producción se deberán manifestar simultáneamente los efectos siguientes:*

- *Síntomas evidentes de daño por efecto mecánico del viento en la vegetación: defoliación*[3] *causada y pelado de las ramas.*

- *Aparición, en los frutos existentes en el árbol, de rezaduras, contusiones, o insiciones.*

- *Existencia de mayores daños en las filas que han estado más expuestas al viento (efecto cortaviento).*

En el supuesto de que, por la ocurrencia del viento con las características anteriormente descritas de produzcan caídas de frutos, estas pérdidas estarán garantizadas únicamente cuando entre los frutos existentes en el suelo, se aprecien frutos con resto del pezón o cáliz. (Agroseguro S.A., 2015)

Una vez aclarado cuando se considera daño por riesgo de Viento, se puede analizar, como indica la Tabla 6, que los valores se concentran en cero, para empezar a crecer a partir del último cuartil, esto representa la naturaleza catastrófica del riesgo, así como un rango de daños de gran varianza y con varios valores atípicos.

[3] Representa la caída de hojas en los árboles, en este caso por el fecto mecánico del viento

Tabla 6. Estadísticos descriptivos de Daños por Viento

DAÑOS (€)		VIENTO
N de datos		408,00
Valores perdidos		0,00
Media		1.815,79
Moda		0,00
Desviación estándar		5.828,09
Varianza		33.970.000
Rango		62.187,4
Mínimo		0
Máximo		62.187,4
Percentiles	**25**	0,00
	50	0,00
	75	1.160,10

Fuente: Elaboración propia

Daños por Helada:

Esta es la variable a explicar del segundo modelo, representa los daños estimados en euros por póliza a causa de heladas.

Dentro de las condiciones generales del seguro para el cultivo de naranaja se define helada como: *Temperatura ambiental igual o inferior a la temperatura crítica mínima de cada una de las fases del desarrollo vegetativo que, debido a la formación de hielo en los tejidos, ocasione una pérdida en los bienes asegurados, con alguno de los efectos siguientes:*

- *Caída del fruto*

- *Necrosis[4] en cualquier órgano de la planta*

- *Detención irreversible del desarrollo de todo o parte del fruto, siempre que venga acompañada de alguna alteración de sus características internas o externas, tales como:*

    a) *Aparición de manchas en la piel o flavedo. No obstante, este síntoma por si solo en el tangelo Fortune no presupone la existencia de helada.*

    b) *Presencia en el fruto de cristales de hesperidina[5].*

    c) *Separación anormal de las membranas de los gajos, llegando incluso a la aparición de cavernas.*

    d) *Aparición de manchas oscuras en las membranas de los gajos.*

    e) *Pérdida de peso por desecación del fruto.*

Muerte del árbol o pérdida de la cosecha del año siguiente por necrosis total o parcial de su órgano vegetativo. (Agroseguro S.A., 2015)

Una vez aclarado la definición de daño por Helada, se describen los principales estadísticos descriptivos para la serie Daños por Helada. Como indica la Tabla 7, se encuentran más concentrados valor, con relación al viento, la media es superior pero el máximo no, siendo menos dispersa y en general con menos atípicos.

[4] Muerte del árbol a causa de una lesión severa que impide la regeneración de sus células
[5] La Hesperdina es uno de los componentes del zumo de los cítricos,

Tabla 7. Estadísticos descriptivos de Daños por Helada

DAÑOS (€)		**HELADA**
N de datos		1.075
Valores perdidos		0
Media		2.093,338
Moda		0
Desviación estándar		4.053,857
Varianza		16.400.000
Rango		44.884,5
Mínimo		0
Máximo		44.884,5
Percentiles	**25**	0
	50	707,125
	75	2.419,736

Fuente: Elaboración propia

Despues de describir y exponer los objetos a explicar se presenta, a continuación, la metodología empleada para el desarrollo de los modelos.

IV. METODOLOGÍA

IV.1. ORGANIZACIÓN DE LA INFORMACIÓN

Para la implementación de una RNA, la literatura con base empírica recomienda hacer un tratamiento básico de los datos, enfocado al correcto funcionamiento de la red, dado que su escala y coherencia influirán en la obtención de resultados adecuados.

Definición de las variables: Se contemplarán las variables ya descritas con anterioridad.

Agrupación o extracción de características: La definición de la red debe enmarcarse en grupos relativamente homogéneos, para optimizar el proceso, un análisis de clusterización o una agrupación factorial facilitan el trabajo tanto analítica como computacionalmente, en este caso los datos a usar son los correspondientes a un ámbito geográfico, niveles de riesgo relativamente homogéneos y subdivido para los riesgos de Helada y Viento.

Depuración de la base de datos consolidada: Esste paso consiste en revisar y eliminar las variables redundantes o altamente correlacionadas, quitar los datos incompletos; aunque las redes suelen tener un grado de tolerancia al error de la información, este no supera el 10% de fallos sin alterar considerablemente los resultados de la red; y hacer un acercamiento a las magnitudes de las variables y el diseño del planteamiento del problema.

En este apartado es útil hacer un análisis de consistencia de los datos, para evitar la inclusión de redundancia o errores de

transcripción, algunas formas sencillas de realizarlo es mediante histogramas, análisis estadístico multivariante o mediante umbrales para definición de atípicos.

En este sentido se revisó la coherencia de la base, mediante una revisión gráfica y partiendo de una serie de datos más corta que el histórico total del producto, para evitar incoherencias en el acoplo inicial de las bases.

Estandarización de las variables: Ya que las variables pueden tener multiplicidad de magnitudes y estos órdenes pueden influir la magnitud de los pesos, y a su vez el análisis de sensibilidad, dado que el peso de las sinapsis tiene una estrecha relación con la magnitud de la entrada. Para evitarlo, la literatura (Kennedy, Lee, Van Roy, Reed, & Lippman, 1997) sugiere normalizar o estandarizar las variables, de los diferentes métodos existentes se ha empleado el que muestra la Ecuación 21.

Ecuación 21. Estandarización variables continuas

$$X_k^* = \left[\frac{X_k - Min(X)}{Max(X) - Min(X)}\right].[Max(X^*) - Min(X^*)] + Min(X^*)$$

Bajo este procedimiento de estandarizaron las variables Superficie asegurada, Valor de la producción, Precio y los Daños por parcela para los riesgos citados.

Codificación de variables categóricas: Por lo general la implementación econométrica de las variables categóricas implica la

inclusión de variables ficticias (variables dummy) que asumen valor 1 si se pertenece a esa categoría, ó 0 en caso contrario. No obstante, durante la revisión bibliográfica se encontró una acérrima recomendación sobre la codificación mediante -1 / +1, aludiendo a que el valor de la actualización de los pesos depende del valor de las entradas, lo que en el momento de la corrección evitaría la mejora del peso. De esta manera se codificaron las variables Variedad (22 categorías) y final de garantía (7 categorías). En una acercamiento al planteamiento del problema se emplearon redes con ficticias 0 - 1, efectivamente, los errores mínimos encontrados fueron muy superiores a las redes estimadas bajo la alternativa -1 / +1.

IV.2. SELECCIÓN DE LA RED

La red base usada fue un Perceptron multicapa, aunque esta sea un aproximador universal, no es posible encontrar la mejor configuración a priori. Sin embargo, se ha demostrado empíricamente, que la mayoría de problemas de pronóstico de índole económica y financiera tienen solución con una sola capa oculta. Por tanto, para solucionar el problema de optimizar la configuración más idónea, se propone un perceptron multicapa base de tres capas (entrada, capa oculta y salida) y diferentes ordenamientos hasta encontrar la respuesta deseada.

Se han dividido las bases en tres grupos Entrenamiento, Validación y Testing, repartidos en 80%, 10% y 10%, respectivamente, la primera fase realiza el aprendizaje, la segunda es usada por Matlab para comprobar sobreajuste y verificar que cambios en los datos producen cambios en las salidas, y la tercera es la de prueba para contrastar el resultado de una red entrenada frente a valores reales.
En la Tabla 8 se pueden ver en los demás parámetros de entrenamiento utilizados.

Tabla 8. Parámetros de entrenamiento

PARÁMETROS	
División de los datos	Aleatorio
Algoritmo de entrenamiento	Levenber-Marquardt
Performance	Error cuadrático medio
Iteraciones	1000
Tiempo	Indeterminado
Gradiente mínimo	1e-05
Gradiente máximo	1e2
Mu mínimo (tasa de aprendizaje)	6e-2

Fuente: Elaboración propia

El algoritmo de entrenamiento fue seleccionado por su capacidad de convergencia, el referente del error cuadrático medio para el Performance se utilizó por su popularidad y fácil comprensión. En cuanto a los parámetros de cotas del gradiente y la tasa de aprendizaje, se emplearon los recomendados por matlab, si bien no son los óptimos, si representan los valores intermedios entre velocidad de aprendizaje y carga computacional.

Definición de la arquitectura base

Como ya se ha mencionado y teniendo en cuenta la imposibilidad de definir una RNA óptima a priori, se ha diseñado un código en Matlab que permite probar diferentes redes. Para ello, se ha partido de la red base configurada con una capa oculta, que tiene la propiedad de variar el número de neuronas desde la mitad hasta el doble del númro de variables a considerar (esto incluye cada una de las variables ficticias creadas para la inclusión de las variables categóricas).

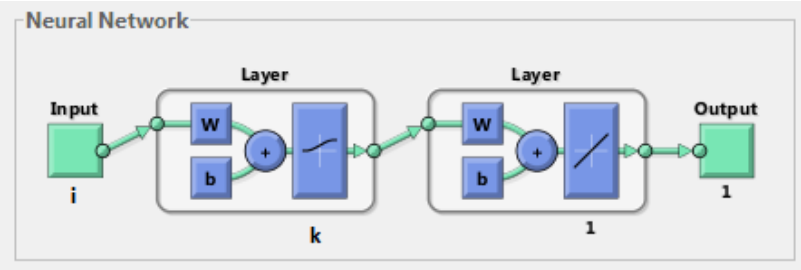
El programa ha sido estructurado para generar el error cuadrático medio de entrenamiento, validación y testing, calcula el R^2 de una regresión cuyos valores de entrada son los resultados de la red (outputs) y la variable a explicar los valores objetivos (targets). Para entrenamiento, validación y testing también calcula el índice relativo de importancia, mencionado en el capítulo anterior, para cada una de las redes programadas.

El código permite configurar El programa también los diferentes parámetros de entrenamiento, función de activación, tasa de aprendizaje, número de iteraciones máximas sin optimizar el error, porcentaje de información a usar en entrenamiento y validación ya mencionados.

En cuanto a la topología de las redes se proponeusar una función de activación logarítmica sigmoidal para la capa oculta, porque genera valores entre 0 y 1, además, en las pruebas previas a la definición de la metodología fue la que mejores resultados aproximó.

Por otro lado acorde con las recomendaciones recopiladas en la revisión bibliográfica, se adopta como función de salida una función de transformación lineal, con la configuración que se presenta en la Ilustración 5. **Arquitectura de la RNA base**, donde *i* representa el número de variables y *k* el número de neuronas.

Ilustración 5. Arquitectura de la RNA base



Fuente: Elaboración propia

Para inicializar los pesos de las sinapsis y agilizar un poco la convergencia del algoritmo se empleó la función *initlay*, la cual utiliza un aleatorio para generar los pesos iniciales. Esto supone un inconveniente, ya que cada vez que Matlab entrena una red, así sea de características similares, no arroja los mismos resultados,.Siendo uno de los mayores inconvenientes a la hora de optimizar el diseño de estos modelos, a continuación se detalla la técnica empleada para (subsanarlo, minimizarlo, lo que prefieras).

IV.3. DEPURACIÓN

Para dar una alternativa al problema mencionado en el apartado anterior, se iteró 10 veces cada uno de los códigos por riesgo y se probaron las redes cuya configuración convergía a los mejores resultados, para estimar 100 conjuntos de RNA con las características mencionadas.

A cada grupo de estimaciones se le genera un ranking bajo los criterios de mayor R^2 en la etapa de testing, pero teniendo un R^2 mayor en la fase de entrenamiento (esto para disminuir la posibilidad de obtener una red sobre entrenada), considerando además la minimización del error cuadrático medio.

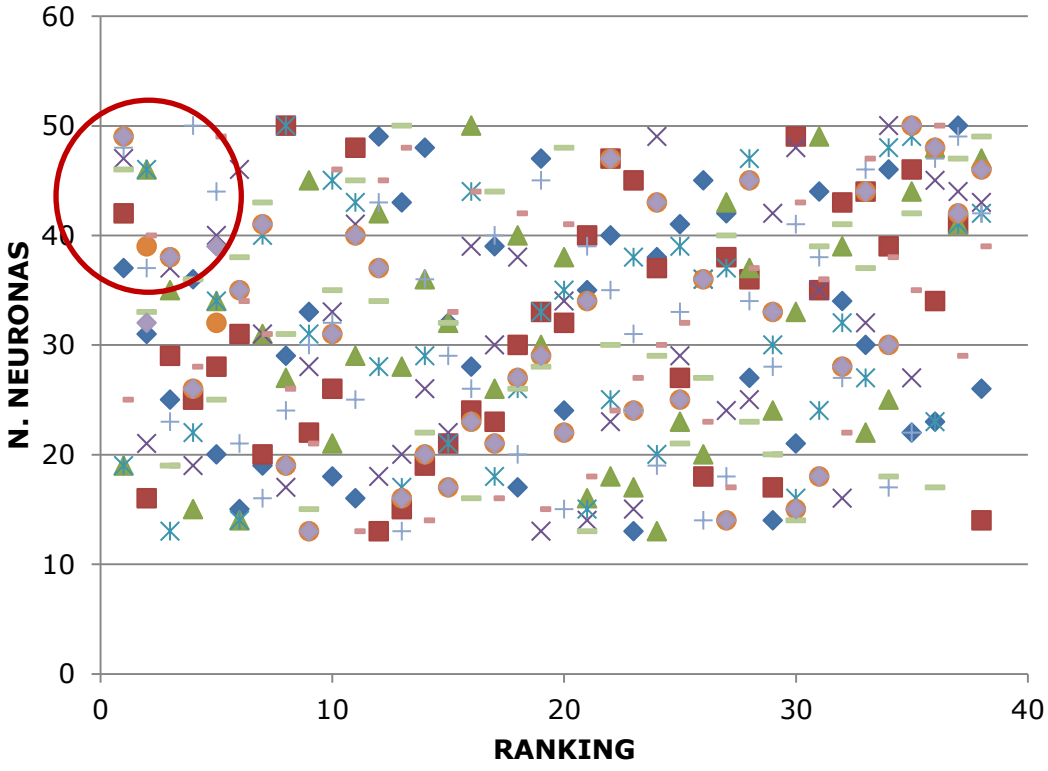
Una vez realizado el ranking se realiza una gráfica para la determinación del número de neuronas más común dentro de las primeras posiciones y dentro de este subgrupo se elige aquella cuyo R^2 en la etapa de testing es mayor. A la red elegida se le realiza el análisis de pesos para valorar la configuración interna de la RNA y se contrasta dicho resultado con la información técnica resultante de la experiencia del seguro.

V. RESULTADOS

V.1. VIENTO

Se realizaron 100 iteraciones del código para generar 38 redes perceptron multicapa, variando de 13 a 50 neuronas, cada estimación se ordenó según el menor error cuadrático medio (mse), con los datos repartidos de forma aleatoria en los grupos de entrenamiento (326), validación (41) y prueba (41), como indica el Gráfico 1 la mejores se encuentran en redes de entre 30 y 50 neuronas.

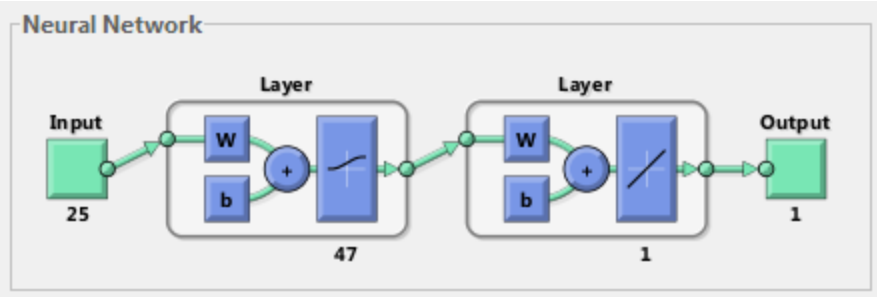
Gráfico 1. Ranking de RNA Viento según menor mse



Fuente: Elaboración propia

El grupo estaba compuesto por 21 RNA, dentro de estas se eligió aquella con el mayor R^2 en la etapa de testing, pero teniendo un R^2 mayor en la fase de entrenamiento, dando como resultado la red con las características de la Ilustración 6.

Ilustración 6. Arquitectura RNA Viento



Fuente: Elaboración propia

Los criterios de selección muestran una RNA con un alto poder predictivo respecto al target, es de resaltar que pese a existir redes con mejores resultados en la etapa de testing, no cumplían con el requisito de superioridad en la etapa de entrenamiento, los resultados de R^2 y mse por etapa se muestran en la Tabla 9.

Tabla 9. Mejor RNA Viento

N.Neuronas	R^2(T)	R^2(V)	R^2(TS)	MSE(T)	MSE(V)	MSE(TS)
47	88,3%	78,7%	80,01%	0,000138	0,00046	0,00028

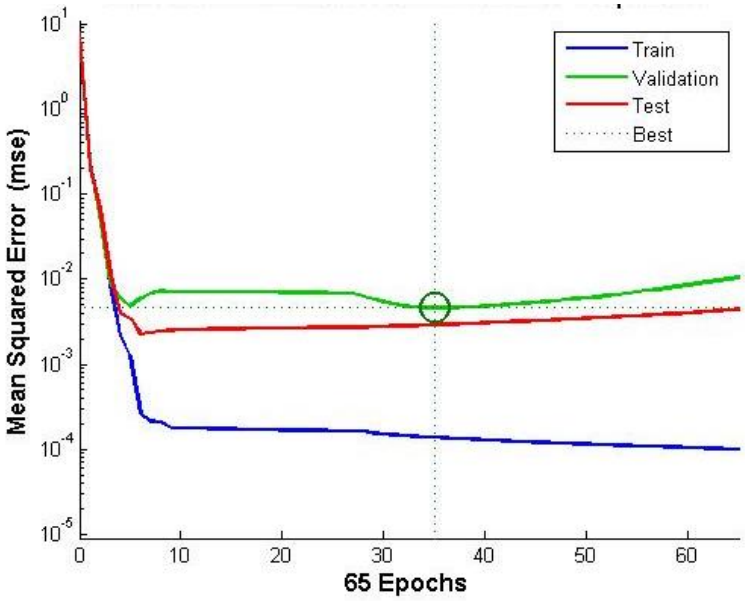
Fuente: Elaboración propia

Es importante mencionar que ninguna red supero las 130 iteraciones, en este caso en particular se realizaron 65, aunque la mejor iteración, como ilustra el Gráfico 2, y de la cual se emplean

los pesos y bias[6], fueron los generados en la iteración 35, las 30 iteraciones adicionales son una medida para evitar caer en mínimos locales.

Hubo una rápida convergenvia hacia el mínimo error cuadrático medio hasta la iteración 6, está presentó para el caso de validación un mínimo local aunque dio paso, en posteriores iteraciones, al mínimo finalmente encontrado.

Gráfico 2. Error cuadrático medio por iteración Viento



Fuente: Elaboración propia

Una vez realizada la red se transformaron los valores de salida el total de daños es de 130.995,13€ y el estimado 134.675,84€, con una diferencia del 2,8% por encima del valor de daños ocurridos.

[6] Ver los pesos en el Anexo2. Pesos Viento

En cuanto al análisis de importancia de las variables cuyos resultados se recogen en la Tabla 10, más allá de la importancia del Valor de producción y la Superficie, que en sí mismos condicionan la dimensión del daño, destaca la importancia de la variedad, reafirmando la necesidad de generar tarifas acorde a la variedad asegurada. Por otro lado la poca importancia del final de garantía corta con la idea de la condicionalidad del daño que suele ser de mayor intensidad según la época del año.

Tabla 10. Análisis de importancia de las variables para la RNA de Viento

Variables	Q	Importancia relativa
Final Garantía	0,039	5,8%
Variedad	0,056	8,2%
Superficie	0,179	26,1%
Valor de Producción	0,685	100%
Precio	0,04	5,8%

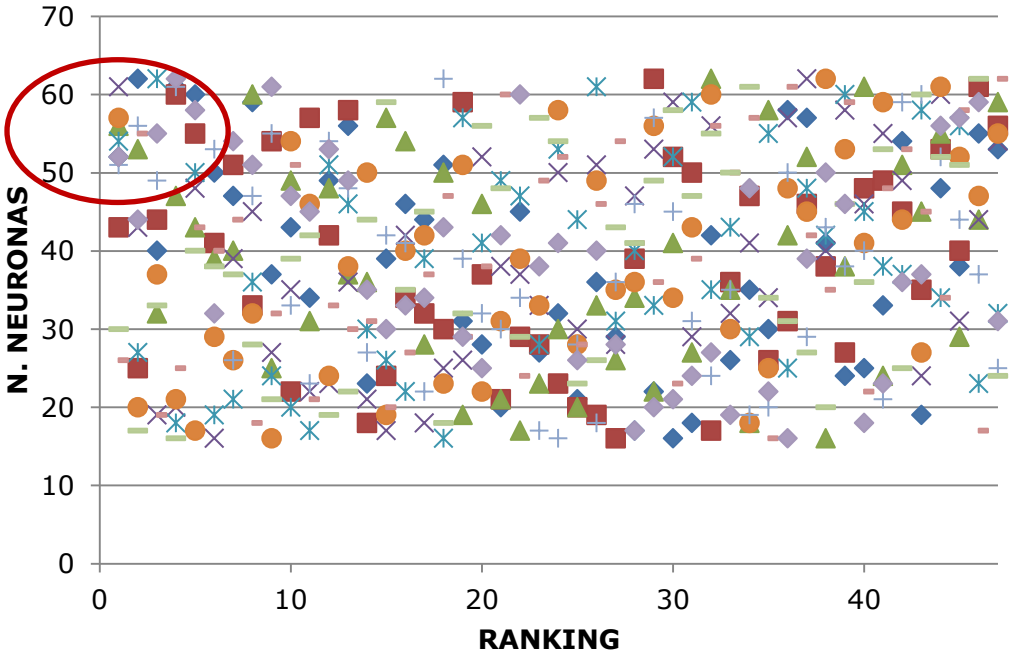
Fuente: Elaboración propia

Esto podría reflejar dos cosas, en primer lugar que las cobertura tomadas para el riesgo en los finales de garantía más críticos está siendo mitigada por externalidades no incluidas en el modelo o que los finales de garantía no contienen una proporción marcada del riesgo si no que en realidad se encuentra a los meses comunes a las opciones (otoño), surgiendo así la posibilidad de no realizar una gran diferenciación en las tarifas para este riesgo según el periodo de cobertura.

V.2. HELADA

Se realizaron 100 iteraciones del código para generar 38 redes perceptron multicapa, variando de 16 a 62 neuronas, cada estimación se ordenó según el menor error cuadrático medio (mse). Como indica el Gráfico 3 las mejores estimaciones se encuentran en redes de entre 50 y 60 neuronas.

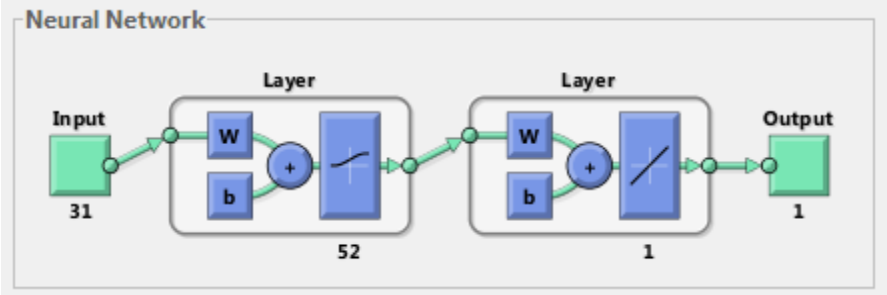
Gráfico 3. Ranking de RNA Viento según menor mse, para las mejores 10 iteraciones Helada



Fuente: Elaboración propia

El grupo estaba compuesto por 19 RNA, dentro de estas se eligió aquella con el mayor R^2 en la etapa de testing, pero teniendo un R^2 mayor en la fase de entrenamiento, dando como resultado la red con las características de la Ilustración 7, de 52 neuronas en la capa oculta y la topología de la RNA base empleada.

Ilustración 7. Arquitectura RNA Helada



Fuente: Elaboración propia

Los criterios de selección muestran una RNA con un alto poder predictivo respecto al target, es de resaltar que pese a existir redes con mejores resultados en la etapa de testing, no cumplían con el requisito de superioridad en la etapa de entrenamiento, los resultados de R^2 y mse por etapa se muestran en la Tabla 9.

Tabla 11. Mejor RNA Helada

N.Neuronas	R^2(T)	R^2(V)	R^2(TS)	MSE(T)	MSE(V)	MSE(TS)
53	65,5%	61,3%	64,53%	0,000014	0,000031	0,000086

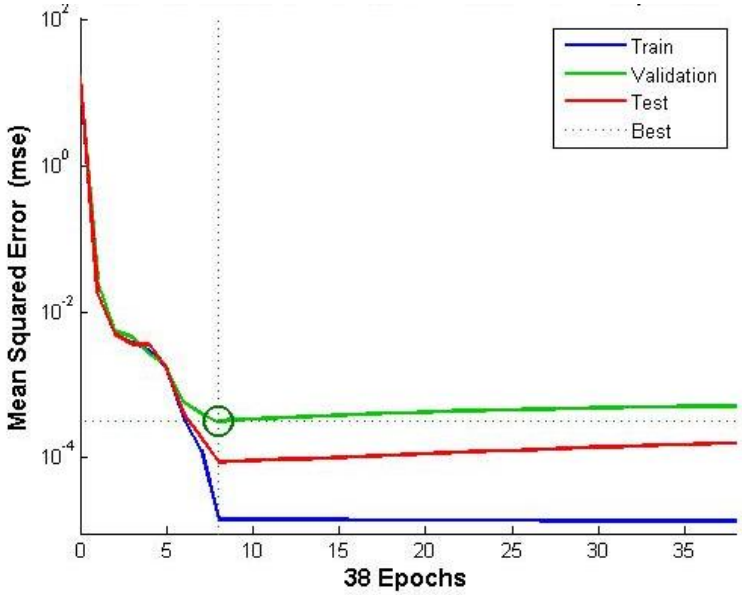
Fuente: Elaboración propia

Es relevante comentar que ninguna red supero las 92 iteraciones, en este caso en particular se realizaron 38, aunque la mejor iteración y de la cual se emplean los pesos y bias[7], fueron los generados en la iteración 8, las 30 iteraciones adicionales son una medida para evitar caer en mínimos locales.

[7] Ver los pesos en el Anexo3. Pesos Helada

Como refleja la Gráfica 4 se encontró un mínimo local para la validación y el testing en la cuarta iteración, después de una rápida convergencia a la minimización del error, sin embargo este fue superado gracias al entrenamiento adicional programado.

Gráfico 4. Error cuadrático medio por iteración RNA Helada



Fuente: Elaboración propia

Una vez realizada la red, se transformaron los valores de salida para obtener un valor estimado de 178.697,92€, frente a un total de daños reales de 174.687,09€. Por tanto, los daños obtenidos en la simulación arrojan con una diferencia del 2,30% por encima del valor de daños realmente ocurridos.

Respecto al análisis de importancia relativa de las variables, presentado en la Tabla 12, se refleja la influencia de una base enriquecida y el número de inputs incluidos, pues en contraste las variables tienen un peso relativo mayor, en especial la variable precio adquiere más importancia que la Variedad, esta relación permite ver

un poco más allá y sugiere empezar a emplear alguna política que en el caso de los daños por Helada de un poco más de valor al precio que a la variedad.

Tabla 12. Análisis de importancia de las variables para la RNA de Helada

Variables	Q	Importancia relativa
Final Garantía	0,058	14,40%
Variedad	0,095	23,70%
Superficie	0,313	78,00%
Valor de Producción	0,401	100,00%
Precio	0,133	33,00%

Fuente: Elaboración propia

Nuevamente se refleja la poca importancia del final de garantía, aunque con una participación relativa mayor, lo que estaría un poco más cerca del análisis intuitivo, puesto que la cobertura a diciembre contiene menor riesgo de helada, mientas el final de garantía que termina en mayo lleva implicta toda la temporada de riesgo.

VI. CONCLUSIONES

Una de las principales limitaciones de este tipo de modelos es la complejidad en su diseño, es por eso que se debe incorporar gran cantidad de posibilidades o iteraciones con criterios de decisión para no sesgar el resultado. Usar pesos de inicialización aleatorios y no repetir la red nos puede hacer caer en un error de omisión de óptimos- Este problema puede solucionarse con la inclusión de modelos de aprendizaje para la optimización de parámetros, o mediante la alternativa propuesta que ha permitido encontrar grupos de RNA con características aceptables y dentro de estos RNA útiles.

Dadas las limitaciones computacionales es necesario compartimentar la red mediante la elaboración degrupos cercanos y pequeños de acuerdo a la capacidad computacional que se posea. Además es importante considerar la relevancia de la inclusión de variables categóricas, ya que dependiendo del número de variables a incluir el requerimiento computacional para el procesamiento crecerá exponencialmente.

Aunque son considerables las diferencias en términos de R^2 de los dos modelos propuestos, Helada y Viento, en términos reales la estimación presenta mejores resultados para el modelo de Helada, siendo más consecuente con el mse que con el R^2. Esto refleja mayor variabilidad en la estimación puntual lo que en el agregado es disipado, obteniendo así en términos de daños globales a pagar una aproximación muy cercana en ambos modelos.

Para el riesgo de helada, el análisis de pesos refleja mayor importancia relativa de las variables incluidas en el modelo, lo que podría conducir a resaltar la importancia de una base de datos con gran nivel de detalle,

tanto cuantitativa como cualitativamente, sin embargo esta configuración no parece afectar la consistencia del pronóstico realizado.

En general los dos modelos proponen estimaciones cercanas en términos agregados, sirviendo de herramienta alternativa a la estimación de los modelos actuariales convencionales, con resultados estimados agregados muy cercanos a los daños reales.

Aunque las modelos de este tipo suelen ser robustos frente a los errores, es importante tener una base de datos depurada, ya que puede generarse mucho ruido y en especial elevar el costo computacional a la hora de converger a un resultado óptimo. Es de vital importancia conocer el problema que se plantea y realizar análisis estadísticos previos al comportamiento de las variables, esto ayuda no solo a tener una base de datos concisa y aclarar la definición del problema, sino que además facilita la definición de la arquitectura y en la medida de lo posible la ponderación inicial de pesos.

Respecto al posible contraste a realizar frente a las tarifas actuales del seguro, se debe mencionar que los resultados aportan una nueva visión sobre la importancia del precio, en el caso de la Helada, y en el caso del Viento, el reducido aporte al modelo del Final de garantías. Se ha generando, por tanto, una herramienta con un potencial uso comercial para el ajuste de las tarifas o que permite hacer nuevos aportes en el análisis de las estimaciones de riesgos y contratación.

En el marco del objetivo general de este trabajo, que pretendía encontrar en los modelos de Redes Neuronales Artificiales una manera alternativa de estimar el daño esperado por póliza y en agregado, se puede concluir que con esta nueva aplicación al Seguro de Naranja, se puede tener una

herramienta consistente y práctica para la obtención de las estimaciones mencionadas, así como un análisis de mayor amplitud hacia las variables a emplear y la consistencia de los resultados obtenidos, facilitando además un acercamiento al manejo comercial de las variables de mayor peso en los modelos de pronósticos de los riesgos.

VII. DESARROLLOS A FUTURO

El trabajo propuesto es completamente extensible a toda la producción de naranja el cultivo y ámbito geográfico, siempre que se cuente con herramientas adecuadas, tanto software como hardware, que permitan optimizar los tiempos de computación y diseño, bien sea mediante el uso de algoritmos genéticos o mapas de vectores o bien mediante la simple ampliación del trabajo ya propuesto a otras zonas del ámbito geográfico.

Por otro lado sería de suma importancia la realización de investigación en pro de la optimización de los tiempos de cómputo y diseño mediante algoritmos genéricos, maquinas de vectores, diseño de corrección de sistemas dinámicos, entre muchas otras opciones, que faciliten la utilización práctica de las RNA.

Con una visión a más largo plazo, podría desarrollarse para otros productos y riesgos ya quem una vez se ha establecido su potencial, debería ser aprovechado en análisis de demanda o clusterización de zonas por intensidad de riesgo mediante mapas auto organizativos, estimación de cambios de la demanda por sensibilidad de las variables, entre otras muchas que podrían ser desarrollados con otros tipos específicos de RNA.

Finalmente, hay que tener en cuenta que la base teóríca de estos modelos se encuentra en construcción, así como nuevos desarrollos enfocados a generalización de reglas de diseño y optimización de parámetros. Por tanto, es de esperar, que nuevas aplicaciones más específicas al ramo actuaria puedan seguir desarrollándose próximamente.

VIII. REFERENCIAS

Actuarial Considerations ApSTAT technologies. (2003). *Introduction to Neural Networks.* Quebec.

Agroseguro S.A. (2015). Condiciones generales, Seguro de cobertura creciente para explotaciones citrícolas. Madrid, España.

Bishop, C. (1995). *Neural Networks for Pattern Recognition.* Clarendon Press: Oxford.

Bousoño, C., Heras, A., & Piedad, T. (2008). *Factores de riesgo y cálculo de primas mediante técnicas de aprendizaje.* Madrid: Fundación Mapfre.

Fausett, L. (1994). *Fundamentals of Neural Networks: Architectures, Algorithms and Applications.* Prentice Hall.

Francis, L. (2002). Modeling for Fraud: Neural Networks Demystified. *Insurance Fraud Bureau of Massachusetts*.

Frost, F., & Karri, V. (1999). Determining the influence of input parameters on BP neural networks output error using sensitivity analysis. *International conference on computational intelligence and miltimedia applications*, (págs. 45-49). New Delhi.

Garson, D. (1991). Interpreting Neural Networks connection weights. *AI Expert*, 46-51.

Gedeon, T. (1997). Data mining of inputs: analysing magnitude and functional measures. *International Journal of Neural Systems*, 209-218.

Hashem, S. (1997). Optimal Linear Combinations of Neural Networks. *Neural Networks, Volume 10, Issue 4*, 599-614.

Hassoun, M. (1995). *Fundamentals of Artificial Neural Networks.* MIT Press.

Holpfield, J. (1982). Neural Networks and Physical Systems with Emergent Collective Computational Abilities. *Biophysics*, 2554-2558.

Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer Feedforward Networks are Universal Approximators. *Neural Networks,Vol 2*, 359-366.

Hornik, K., Stinchcombe, M., & White, H. (1990). Universal Approximation of an Unknown Mapping and its Derivatives. *Neural Networks, vol 3*, 551-560.

Kennedy, R., Lee, Y., Van Roy, B., Reed, C., & Lippman, R. (1997). *Solving Data Mining Problems Through Pattern Recognition.* Prentice Hall.

Kitchens, F. L. (2005). Neural Networks used in Automobile insurance underwritingNeural Networks used in Automobile insurance underwriting. En M. Khosrow-Pour, *Encyclopedia of Information Science and Technology* (págs. 168-172). Londres: Idea Group Inc.

Kitchens, F., Booker, Q. E., & Rebman, C. (2005). *An application of neural networks to insurance underwriting.* Muncie.

Kung, S. (1993). *Digital Naural Networks.* Prentice-Hall.

Le Cun, Y., Denker, J., & Solla, S. (1990). Optimal Brain Damage. En M. Kaufmann, *Advances in Neural Information Processing* (págs. 598-605). San Mateo: Touretzjy.

Lin, C.-T., & Lee, G. (1996). *Neural Fuzzy Systems: A Neuro-Fuzzy Synergism to Intelligent Systems.* Prentic-Hall.

Lisboa, P. M. (1994). The interpretation of supervised neural networks. *Proceedings of the Workshop on Neural Network* (págs. 11-17). Los Alamitos: Taylor.

McCulloch, W., & Pitts, W. (1943). A logical calculus of Ideas Imminent in Nervous Activity. *Bulletin of Mathematical Biophysics 9*, 127-147.

Nath, R., Rajagopalan, B., & Ryker, R. (1997). Determining the saliency of input variables in neural networks classifiers. *Journal of computers*, 767-773.

Orozco Bedoya, V. (2014). *Modelo anulación de pólizas a través de redes neuronales y modelo LOGIT.* Madrid.

Pelessoni, R., & Picech, L. (2002). *An application of unsupervised neural networks in general insurance: the determination of tariff classes.* Trieste.

Purves, D., Fitzpatrick, D., Hall, E., Mc Namara, J., & Williams, M. (2007). *Neurociencia.* Bogotá: Panamericana.

Ricote Gil, F. (2003). *TEORÍA DE LAS REDES NEURONALES PARA LA GERENCIA DE RIESGOS.* Madrid.

Sánchez, D. (2013). *Networks, Longevity Model using Artificial Neural.* Madrid.

Scarselli, F., & Chung Tsoi, A. (1998). Universal Approximation Using Feedforward. *Neural Networks, vol 11*, 15-37.

Shapiro, A. F. (2000). *Soft Computing Applicationsin Actuarial Science.* Penn: Penn State University.

Tanco , F. (2000). *Introducción a las redes neuronales.* Buenos Aires: UNIVERSIDAD TECNOLOGICA NACIONAL.

Zurada, J. M., Malinowski, A., & Cloete, I. (1994). Sensitivity Analysis for Minimization of Input Data Dimension for Feedforward Neural Networks. *IEEE*, 447-450.

IX. ANEXOS

ANEXO 1. CÓDIGO

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%función de anidados para generar 10 grupos de iteraciones
function
[red,Qdos,Q,R2T,R2V,R2TS,MSEV,MSET]=HELADA(H_In11,H_target)

    target = H_target.mat;
    inputsH = H_In11.mat;
    k=size(inputsH);
    k=k(1);% numero de variables
    a=1;
    for n = 1:10
        switch n
            case 1
                 b=1;

                    for l= round(k/2):1:(k*2) % numero de neuronas
                        disp([(n),(b),(k)])
                       %%%%%%%%%%%%%%
                       % CREAR RNN %
                       %%%%%%%%%%%%%%
                       %ARQUITECTURA
                        net = newff(minmax(inputsH),[l 1],{'logsig'
'purelin'},'trainlm');
                        %[neuronas de entrada y salida],
                        %{funciones de activación logarítmica
sigmoidal y lineal}
                        % Algortimo de aprendizaje
                        net.performFcn = 'mse';% error cuadrático
medio

                       %DIVISIÓN DE DATPS

                        net.divideFcn = 'dividerand';  % División
aleatoria
                        net.divideMode = 'sample';  % Divide up
every sample
                        net.divideParam.trainRatio = 80/100;
                        net.divideParam.valRatio = 10/100;
                        net.divideParam.testRatio = 10/100;

                        %PARÁMETROS DE ENTRENAMIENTO
                        net.trainParam.epochs = 1000; %Número
máximo de iteraciones
                        net.trainParam.goal = 0; %Error objetivo
                        net.trainParam.time = inf;%tiempo máximo de
entrenamiento
                        net.trainParam.min_grad = 1e-5;%Gradiente
mínimo
                        net.trainParam.lr = 0.06;%Mu tasa de
aprendizaje
```

```
                        net.trainParam.max_fail=30;% Iteraciones
despues de encontrar un mínimo

                        net = initlay(net);%inicialización de pesos


                        % ENTRENAMIENTO DE LA RED
                       %%%%%
                       [net,tr] = train(net,inputsH,target);

                       %GeneraciOn de pesos
                       %%%%
                       PesoEntradasF= net.IW;
                       PesosCapasF = net.LW;
                       PesosBiasF=net.b; %representa la constante


PesosCapas1=abs(cell2mat(PesoEntradasF(1,1)));

Pesos_Entrada(a,b)=struct('pesos',(cell2mat(PesoEntradasF(1,1))));

                       PesosCapas2=abs(cell2mat(PesosCapasF(2,1)));

Pesos_Salida(a,b)=struct('pesos',(cell2mat(PesosCapasF(2,1))));


PesosBias(a,b)=struct('pesos',(cell2mat(PesosBiasF(1,1))));


                       %Q(im) peso de la variable i a la salida m,
                       %Guardado en un arreglo matricial por
iteración

                        for I=1:1:k
                        num1=0;%numerador de la ecuaciOn en 0 para
acumular y generar la sumatoria
                        den2=0;%denominador de la ecuaciOn en 0
para acumular y generar la sumatoria
                            for i=1:1:k
                                 for j=1:1:l
                                      O=sum(PesosCapas1');  %O es
un vector sumatoria r,j

num1=(PesosCapas1(j,I)./O(j)).*PesosCapas2(j)+num1;

den2=(PesosCapas1(j,i)./O(j)).*PesosCapas2(j)+den2;
                                      Q(a,b,I)=num1./den2;
                                 end
                            end
                        end


                        format short %Banderas creadas para hacer
seguimiento de las iteraciones en caso de ser necesario

                        red(a,b,l) =
struct('cod',((10000)+(k*100)+(l)),'dir',tr);
```

```
                        %DEFINICIÓN DE VARIABLES PARA ANÁLISIS DE
CORRELACIÓN Y ERRORES
                        %%%%%%%%
                        output = net(inputsH);
                        trOut = output(tr.trainInd);
                        vOut = output(tr.valInd);
                        tsOut = output(tr.testInd);
                        trTarg = target(tr.trainInd);
                        vTarg = target(tr.valInd);
                        tsTarg = target(tr.testInd);

                         %Regresiones y R^2
                         mdl = LinearModel.fit(trTarg,trOut);
                         T=mdl.Rsquared.Adjusted;
                         R2T(a,b)=T;
                         mdl = LinearModel.fit(vTarg,vOut);
                         V=mdl.Rsquared.Adjusted;
                         R2V(a,b)=V;
                         mdl = LinearModel.fit(tsTarg,tsOut);
                         TS=mdl.Rsquared.Adjusted;
                         R2TS(a,b)=TS;

                        %analisis del error
                        mseT=mean((trOut-trTarg).^2);
                        mseV=mean((vTarg-vOut).^2);
                        mseTS=mean((tsTarg-tsOut).^2);

                        MSET(a,b)= mseT;
                        MSEV(a,b)= mseV;
                        MSETS(a,b)= mseTS;

                        Qdos(a,b) = 1-(mseV/mseT);

                        b=b+1;

                    end

          a=a+1;

          case 2
          disp((n))

                   b=1;

                     for l= round(k/2):1:(k*2) % numero de neuronas
                         disp([(n),(b),(k)])
                        %%%%%%%%%%%%%
                        % CREAR RNN %
                        %%%%%%%%%%%%%
                        %ARQUITECTURA
                         net = newff(minmax(inputsH),[l 1],{'logsig'
'purelin'},'trainlm');
                         %[neuronas de entrada y salida],
                         %{funciones de activación logarítmica
sigmoidal y lineal}
                         % Algortimo de aprendizaje
                         net.performFcn = 'mse';% error cuadrático
medio
```

```
                        %DIVISIÓN DE DATPS

                         net.divideFcn = 'dividerand';  % División
aleatoria
                         net.divideMode = 'sample';  % Divide up
every sample
                         net.divideParam.trainRatio = 80/100;
                         net.divideParam.valRatio = 10/100;
                         net.divideParam.testRatio = 10/100;

                         %PARÁMETROS DE ENTRENAMIENTO
                         net.trainParam.epochs = 1000; %Número
máximo de iteraciones
                         net.trainParam.goal = 0; %Error objetivo
                         net.trainParam.time = inf;%tiempo máximo de
entrenamiento
                         net.trainParam.min_grad = 1e-5;%Gradiente
mínimo
                         net.trainParam.lr = 0.06;%Mu tasa de
aprendizaje
                         net.trainParam.max_fail=30;% Iteraciones
despues de encontrar un mínimo

                         net = initlay(net);%inicialización de pesos


                         % ENTRENAMIENTO DE LA RED
                        %%%%%
                        [net,tr] = train(net,inputsH,target);

                        %GeneraciOn de pesos
                        %%%%
                        PesoEntradasF= net.IW;
                        PesosCapasF = net.LW;
                        PesosBiasF=net.b; %representa la constante


PesosCapas1=abs(cell2mat(PesoEntradasF(1,1)));

Pesos_Entrada(a,b)=struct('pesos',(cell2mat(PesoEntradasF(1,1))));

                        PesosCapas2=abs(cell2mat(PesosCapasF(2,1)));

Pesos_Salida(a,b)=struct('pesos',(cell2mat(PesosCapasF(2,1))));


PesosBias(a,b)=struct('pesos',(cell2mat(PesosBiasF(1,1))));


                        %Q(im) peso de la variable i a la salida m,
                        %Guardado en un arreglo matricial por
iteración

                         for I=1:1:k
                         num1=0;%numerador de la ecuaciOn en 0 para
acumular y generar la sumatoria
                         den2=0;%denominador de la ecuaciOn en 0
para acumular y generar la sumatoria
```

```
                            for i=1:1:k
                                 for j=1:1:l
                                      O=sum(PesosCapas1');  %O es
un vector sumatoria r,j

num1=(PesosCapas1(j,I)./O(j)).*PesosCapas2(j)+num1;

den2=(PesosCapas1(j,i)./O(j)).*PesosCapas2(j)+den2;
                                      Q(a,b,I)=num1./den2;
                                 end
                            end
                        end


                        format short %Banderas creadas para hacer
seguimiento de las iteraciones en caso de ser necesario

                        red(a,b,l) =
struct('cod',((10000)+(k*100)+(l)),'dir',tr);


                        %DEFINICIÓN DE VARIABLES PARA ANÁLISIS DE
CORRELACIÓN Y ERRORES
                        %%%%%%%%
                        output = net(inputsH);
                        trOut = output(tr.trainInd);
                        vOut = output(tr.valInd);
                        tsOut = output(tr.testInd);
                        trTarg = target(tr.trainInd);
                        vTarg = target(tr.valInd);
                        tsTarg = target(tr.testInd);

                         %Regresiones y R^2
                         mdl = LinearModel.fit(trTarg,trOut);
                         T=mdl.Rsquared.Adjusted;
                         R2T(a,b)=T;
                         mdl = LinearModel.fit(vTarg,vOut);
                         V=mdl.Rsquared.Adjusted;
                         R2V(a,b)=V;
                         mdl = LinearModel.fit(tsTarg,tsOut);
                         TS=mdl.Rsquared.Adjusted;
                         R2TS(a,b)=TS;

                        %analisis del error
                        mseT=mean((trOut-trTarg).^2);
                        mseV=mean((vTarg-vOut).^2);
                        mseTS=mean((tsTarg-tsOut).^2);

                        MSET(a,b)= mseT;
                        MSEV(a,b)= mseV;
                        MSETS(a,b)= mseTS;

                        Qdos(a,b) = 1-(mseV/mseT);

                        b=b+1;

                    end
                    a=a+1;
```

```
            case 3
                 b=1;

                    for l= round(k/2):1:(k*2) % numero de neuronas
                        disp([(n),(b),(k)])
                       %%%%%%%%%%%%%
                       % CREAR RNN %
                       %%%%%%%%%%%%%
                       %ARQUITECTURA
                        net = newff(minmax(inputsH),[l 1],{'logsig'
'purelin'},'trainlm');
                        %[neuronas de entrada y salida],
                        %{funciones de activación logarítmica
sigmoidal y lineal}
                        % Algortimo de aprendizaje
                        net.performFcn = 'mse';% error cuadrático
medio

                       %DIVISIÓN DE DATPS

                        net.divideFcn = 'dividerand';  % División
aleatoria
                        net.divideMode = 'sample';  % Divide up
every sample
                        net.divideParam.trainRatio = 80/100;
                        net.divideParam.valRatio = 10/100;
                        net.divideParam.testRatio = 10/100;

                        %PARÁMETROS DE ENTRENAMIENTO
                        net.trainParam.epochs = 1000; %Número
máximo de iteraciones
                        net.trainParam.goal = 0; %Error objetivo
                        net.trainParam.time = inf;%tiempo máximo de
entrenamiento
                        net.trainParam.min_grad = 1e-5;%Gradiente
mínimo
                        net.trainParam.lr = 0.06;%Mu tasa de
aprendizaje
                        net.trainParam.max_fail=30;% Iteraciones
despues de encontrar un mínimo

                        net = initlay(net);%inicialización de pesos


                        % ENTRENAMIENTO DE LA RED
                       %%%%%
                       [net,tr] = train(net,inputsH,target);

                       %GeneraciOn de pesos
                       %%%%
                       PesoEntradasF= net.IW;
                       PesosCapasF = net.LW;
                       PesosBiasF=net.b; %representa la constante


PesosCapas1=abs(cell2mat(PesoEntradasF(1,1)));

Pesos_Entrada(a,b)=struct('pesos',(cell2mat(PesoEntradasF(1,1))));
```

```
                        PesosCapas2=abs(cell2mat(PesosCapasF(2,1)));

Pesos_Salida(a,b)=struct('pesos',(cell2mat(PesosCapasF(2,1))));


PesosBias(a,b)=struct('pesos',(cell2mat(PesosBiasF(1,1))));


                       %Q(im) peso de la variable i a la salida m,
                       %Guardado en un arreglo matricial por
iteración

                        for I=1:1:k
                        num1=0;%numerador de la ecuaciOn en 0 para
acumular y generar la sumatoria
                        den2=0;%denominador de la ecuaciOn en 0
para acumular y generar la sumatoria
                            for i=1:1:k
                                 for j=1:1:l
                                      O=sum(PesosCapas1');  %O es
un vector sumatoria r,j

num1=(PesosCapas1(j,I)./O(j)).*PesosCapas2(j)+num1;

den2=(PesosCapas1(j,i)./O(j)).*PesosCapas2(j)+den2;
                                      Q(a,b,I)=num1./den2;
                                 end
                            end
                        end


                        format short %Banderas creadas para hacer
seguimiento de las iteraciones en caso de ser necesario

                        red(a,b,l) =
struct('cod',((10000)+(k*100)+(l)),'dir',tr);


                        %DEFINICIÓN DE VARIABLES PARA ANÁLISIS DE
CORRELACIÓN Y ERRORES
                        %%%%%%%%
                        output = net(inputsH);
                        trOut = output(tr.trainInd);
                        vOut = output(tr.valInd);
                        tsOut = output(tr.testInd);
                        trTarg = target(tr.trainInd);
                        vTarg = target(tr.valInd);
                        tsTarg = target(tr.testInd);

                         %Regresiones y R^2
                         mdl = LinearModel.fit(trTarg,trOut);
                         T=mdl.Rsquared.Adjusted;
                         R2T(a,b)=T;
                         mdl = LinearModel.fit(vTarg,vOut);
                         V=mdl.Rsquared.Adjusted;
                         R2V(a,b)=V;
                         mdl = LinearModel.fit(tsTarg,tsOut);
                         TS=mdl.Rsquared.Adjusted;
                         R2TS(a,b)=TS;
```

```
                        %analisis del error
                        mseT=mean((trOut-trTarg).^2);
                        mseV=mean((vTarg-vOut).^2);
                        mseTS=mean((tsTarg-tsOut).^2);

                        MSET(a,b)= mseT;
                        MSEV(a,b)= mseV;
                        MSETS(a,b)= mseTS;

                        Qdos(a,b) = 1-(mseV/mseT);

                        b=b+1;

                    end

        a=a+1;


            case 4
                 b=1;

                for l= round(k/2):1:(k*2) % numero de neuronas
                    disp([(n),(b),(k)])
                   %%%%%%%%%%%%%
                   % CREAR RNN %
                   %%%%%%%%%%%%%
                   %ARQUITECTURA
                    net = newff(minmax(inputsH),[l 1],{'logsig'
'purelin'},'trainlm');
                    %[neuronas de entrada y salida],
                    %{funciones de activación logarítmica
sigmoidal y lineal}
                    % Algortimo de aprendizaje
                    net.performFcn = 'mse';% error cuadrático
medio

                   %DIVISIÓN DE DATPS

                    net.divideFcn = 'dividerand';  % División
aleatoria
                    net.divideMode = 'sample';  % Divide up
every sample
                    net.divideParam.trainRatio = 80/100;
                    net.divideParam.valRatio = 10/100;
                    net.divideParam.testRatio = 10/100;

                    %PARÁMETROS DE ENTRENAMIENTO
                    net.trainParam.epochs = 1000; %Número
máximo de iteraciones
                    net.trainParam.goal = 0; %Error objetivo
                    net.trainParam.time = inf;%tiempo máximo de
entrenamiento
                    net.trainParam.min_grad = 1e-5;%Gradiente
mínimo
                    net.trainParam.lr = 0.06;%Mu tasa de
aprendizaje
                    net.trainParam.max_fail=30;% Iteraciones
despues de encontrar un mínimo
```

```
                        net = initlay(net);%inicialización de pesos


                        % ENTRENAMIENTO DE LA RED
                        %%%%%
                        [net,tr] = train(net,inputsH,target);

                        %GeneraciOn de pesos
                        %%%%
                        PesoEntradasF= net.IW;
                        PesosCapasF = net.LW;
                        PesosBiasF=net.b; %representa la constante


PesosCapas1=abs(cell2mat(PesoEntradasF(1,1)));

Pesos_Entrada(a,b)=struct('pesos',(cell2mat(PesoEntradasF(1,1))));

                        PesosCapas2=abs(cell2mat(PesosCapasF(2,1)));

Pesos_Salida(a,b)=struct('pesos',(cell2mat(PesosCapasF(2,1))));


PesosBias(a,b)=struct('pesos',(cell2mat(PesosBiasF(1,1))));


                        %Q(im) peso de la variable i a la salida m,
                        %Guardado en un arreglo matricial por
iteración

                        for I=1:1:k
                        num1=0;%numerador de la ecuaciOn en 0 para
acumular y generar la sumatoria
                        den2=0;%denominador de la ecuaciOn en 0
para acumular y generar la sumatoria
                            for i=1:1:k
                                 for j=1:1:l
                                      O=sum(PesosCapas1');  %O es
un vector sumatoria r,j

num1=(PesosCapas1(j,I)./O(j)).*PesosCapas2(j)+num1;

den2=(PesosCapas1(j,i)./O(j)).*PesosCapas2(j)+den2;
                                      Q(a,b,I)=num1./den2;
                                 end
                            end
                        end


                        format short %Banderas creadas para hacer
seguimiento de las iteraciones en caso de ser necesario

                        red(a,b,l) =
struct('cod',((10000)+(k*100)+(l)),'dir',tr);
```

```
                        %DEFINICIÓN DE VARIABLES PARA ANÁLISIS DE
CORRELACIÓN Y ERRORES
                        %%%%%%%%
                        output = net(inputsH);
                        trOut = output(tr.trainInd);
                        vOut = output(tr.valInd);
                        tsOut = output(tr.testInd);
                        trTarg = target(tr.trainInd);
                        vTarg = target(tr.valInd);
                        tsTarg = target(tr.testInd);

                         %Regresiones y R^2
                         mdl = LinearModel.fit(trTarg,trOut);
                         T=mdl.Rsquared.Adjusted;
                         R2T(a,b)=T;
                         mdl = LinearModel.fit(vTarg,vOut);
                         V=mdl.Rsquared.Adjusted;
                         R2V(a,b)=V;
                         mdl = LinearModel.fit(tsTarg,tsOut);
                         TS=mdl.Rsquared.Adjusted;
                         R2TS(a,b)=TS;

                        %analisis del error
                        mseT=mean((trOut-trTarg).^2);
                        mseV=mean((vTarg-vOut).^2);
                        mseTS=mean((tsTarg-tsOut).^2);

                        MSET(a,b)= mseT;
                        MSEV(a,b)= mseV;
                        MSETS(a,b)= mseTS;

                        Qdos(a,b) = 1-(mseV/mseT);

                        b=b+1;

                    end

          a=a+1;

             case 5
                  b=1;

                    for l= round(k/2):1:(k*2) % numero de neuronas
                        disp([(n),(b),(k)])
                       %%%%%%%%%%%%%
                       % CREAR RNN %
                       %%%%%%%%%%%%%
                       %ARQUITECTURA
                        net = newff(minmax(inputsH),[l 1],{'logsig'
'purelin'},'trainlm');
                        %[neuronas de entrada y salida],
                        %{funciones de activación logarítmica
sigmoidal y lineal}
                        % Algortimo de aprendizaje
                        net.performFcn = 'mse';% error cuadrático
medio

                      %DIVISIÓN DE DATPS
```

```
                        net.divideFcn = 'dividerand';  % División
aleatoria
                        net.divideMode = 'sample';  % Divide up
every sample
                        net.divideParam.trainRatio = 80/100;
                        net.divideParam.valRatio = 10/100;
                        net.divideParam.testRatio = 10/100;

                        %PARÁMETROS DE ENTRENAMIENTO
                        net.trainParam.epochs = 1000; %Número
máximo de iteraciones
                        net.trainParam.goal = 0; %Error objetivo
                        net.trainParam.time = inf;%tiempo máximo de
entrenamiento
                        net.trainParam.min_grad = 1e-5;%Gradiente
mínimo
                        net.trainParam.lr = 0.06;%Mu tasa de
aprendizaje
                        net.trainParam.max_fail=30;% Iteraciones
despues de encontrar un mínimo

                        net = initlay(net);%inicialización de pesos


                         % ENTRENAMIENTO DE LA RED
                        %%%%%
                        [net,tr] = train(net,inputsH,target);

                        %GeneraciOn de pesos
                        %%%%
                        PesoEntradasF= net.IW;
                        PesosCapasF = net.LW;
                        PesosBiasF=net.b; %representa la constante


PesosCapas1=abs(cell2mat(PesoEntradasF(1,1)));

Pesos_Entrada(a,b)=struct('pesos',(cell2mat(PesoEntradasF(1,1))));

                       PesosCapas2=abs(cell2mat(PesosCapasF(2,1)));

Pesos_Salida(a,b)=struct('pesos',(cell2mat(PesosCapasF(2,1))));


PesosBias(a,b)=struct('pesos',(cell2mat(PesosBiasF(1,1))));


                        %Q(im) peso de la variable i a la salida m,
                        %Guardado en un arreglo matricial por
iteración

                         for I=1:1:k
                         num1=0;%numerador de la ecuaciOn en 0 para
acumular y generar la sumatoria
                         den2=0;%denominador de la ecuaciOn en 0
para acumular y generar la sumatoria
                             for i=1:1:k
                                  for j=1:1:l
```

```
                                        O=sum(PesosCapas1');  %O es
un vector sumatoria r,j

num1=(PesosCapas1(j,I)./O(j)).*PesosCapas2(j)+num1;

den2=(PesosCapas1(j,i)./O(j)).*PesosCapas2(j)+den2;
                                        Q(a,b,I)=num1./den2;
                                    end
                                end
                            end


                            format short %Banderas creadas para hacer
seguimiento de las iteraciones en caso de ser necesario

                            red(a,b,l) =
struct('cod',((10000)+(k*100)+(l)),'dir',tr);


                            %DEFINICIÓN DE VARIABLES PARA ANÁLISIS DE
CORRELACIÓN Y ERRORES
                            %%%%%%%%
                            output = net(inputsH);
                            trOut = output(tr.trainInd);
                            vOut = output(tr.valInd);
                            tsOut = output(tr.testInd);
                            trTarg = target(tr.trainInd);
                            vTarg = target(tr.valInd);
                            tsTarg = target(tr.testInd);

                             %Regresiones y R^2
                             mdl = LinearModel.fit(trTarg,trOut);
                             T=mdl.Rsquared.Adjusted;
                             R2T(a,b)=T;
                             mdl = LinearModel.fit(vTarg,vOut);
                             V=mdl.Rsquared.Adjusted;
                             R2V(a,b)=V;
                             mdl = LinearModel.fit(tsTarg,tsOut);
                             TS=mdl.Rsquared.Adjusted;
                             R2TS(a,b)=TS;

                            %analisis del error
                            mseT=mean((trOut-trTarg).^2);
                            mseV=mean((vTarg-vOut).^2);
                            mseTS=mean((tsTarg-tsOut).^2);

                            MSET(a,b)= mseT;
                            MSEV(a,b)= mseV;
                            MSETS(a,b)= mseTS;

                            Qdos(a,b) = 1-(mseV/mseT);

                            b=b+1;

                        end

                a=a+1;

                    case 6
```

```
                b=1;

                    for l= round(k/2):1:(k*2) % numero de neuronas
                        disp([(n),(b),(k)])
                        %%%%%%%%%%%%%
                        % CREAR RNN %
                        %%%%%%%%%%%%%
                        %ARQUITECTURA
                        net = newff(minmax(inputsH),[l 1],{'logsig'
'purelin'},'trainlm');
                        %[neuronas de entrada y salida],
                        %{funciones de activación logarítmica
sigmoidal y lineal}
                        % Algortimo de aprendizaje
                        net.performFcn = 'mse';% error cuadrático
medio

                        %DIVISIÓN DE DATPS

                        net.divideFcn = 'dividerand';  % División
aleatoria
                        net.divideMode = 'sample';  % Divide up
every sample
                        net.divideParam.trainRatio = 80/100;
                        net.divideParam.valRatio = 10/100;
                        net.divideParam.testRatio = 10/100;

                        %PARÁMETROS DE ENTRENAMIENTO
                        net.trainParam.epochs = 1000; %Número
máximo de iteraciones
                        net.trainParam.goal = 0; %Error objetivo
                        net.trainParam.time = inf;%tiempo máximo de
entrenamiento
                        net.trainParam.min_grad = 1e-5;%Gradiente
mínimo
                        net.trainParam.lr = 0.06;%Mu tasa de
aprendizaje
                        net.trainParam.max_fail=30;% Iteraciones
despues de encontrar un mínimo

                        net = initlay(net);%inicialización de pesos


                        % ENTRENAMIENTO DE LA RED
                        %%%%%
                        [net,tr] = train(net,inputsH,target);

                        %GeneraciOn de pesos
                        %%%%
                        PesoEntradasF= net.IW;
                        PesosCapasF = net.LW;
                        PesosBiasF=net.b; %representa la constante


PesosCapas1=abs(cell2mat(PesoEntradasF(1,1)));

Pesos_Entrada(a,b)=struct('pesos',(cell2mat(PesoEntradasF(1,1))));

                        PesosCapas2=abs(cell2mat(PesosCapasF(2,1)));
```

```
Pesos_Salida(a,b)=struct('pesos',(cell2mat(PesosCapasF(2,1))));


PesosBias(a,b)=struct('pesos',(cell2mat(PesosBiasF(1,1))));


                        %Q(im) peso de la variable i a la salida m,
                        %Guardado en un arreglo matricial por
iteración

                         for I=1:1:k
                         num1=0;%numerador de la ecuaciOn en 0 para
acumular y generar la sumatoria
                         den2=0;%denominador de la ecuaciOn en 0
para acumular y generar la sumatoria
                             for i=1:1:k
                                  for j=1:1:l
                                       O=sum(PesosCapas1');  %O es
un vector sumatoria r,j

num1=(PesosCapas1(j,I)./O(j)).*PesosCapas2(j)+num1;

den2=(PesosCapas1(j,i)./O(j)).*PesosCapas2(j)+den2;
                                       Q(a,b,I)=num1./den2;
                                  end
                             end
                         end


                         format short %Banderas creadas para hacer
seguimiento de las iteraciones en caso de ser necesario

                         red(a,b,l) =
struct('cod',((10000)+(k*100)+(l)),'dir',tr);


                         %DEFINICIÓN DE VARIABLES PARA ANÁLISIS DE
CORRELACIÓN Y ERRORES
                         %%%%%%%%
                         output = net(inputsH);
                         trOut = output(tr.trainInd);
                         vOut = output(tr.valInd);
                         tsOut = output(tr.testInd);
                         trTarg = target(tr.trainInd);
                         vTarg = target(tr.valInd);
                         tsTarg = target(tr.testInd);

                          %Regresiones y R^2
                          mdl = LinearModel.fit(trTarg,trOut);
                          T=mdl.Rsquared.Adjusted;
                          R2T(a,b)=T;
                          mdl = LinearModel.fit(vTarg,vOut);
                          V=mdl.Rsquared.Adjusted;
                          R2V(a,b)=V;
                          mdl = LinearModel.fit(tsTarg,tsOut);
                          TS=mdl.Rsquared.Adjusted;
                          R2TS(a,b)=TS;
```

```
                        %analisis del error
                        mseT=mean((trOut-trTarg).^2);
                        mseV=mean((vTarg-vOut).^2);
                        mseTS=mean((tsTarg-tsOut).^2);

                        MSET(a,b)= mseT;
                        MSEV(a,b)= mseV;
                        MSETS(a,b)= mseTS;

                        Qdos(a,b) = 1-(mseV/mseT);

                        b=b+1;

                    end

          a=a+1;

               case 7
                  b=1;

                    for l= round(k/2):1:(k*2) % numero de neuronas
                        disp([(n),(b),(k)])
                       %%%%%%%%%%%%%
                       % CREAR RNN %
                       %%%%%%%%%%%%%
                       %ARQUITECTURA
                        net = newff(minmax(inputsH),[l 1],{'logsig'
'purelin'},'trainlm');
                        %[neuronas de entrada y salida],
                        %{funciones de activación logarítmica
sigmoidal y lineal}
                        % Algortimo de aprendizaje
                        net.performFcn = 'mse';% error cuadrático
medio

                       %DIVISIÓN DE DATPS

                        net.divideFcn = 'dividerand';  % División
aleatoria
                        net.divideMode = 'sample';  % Divide up
every sample
                        net.divideParam.trainRatio = 80/100;
                        net.divideParam.valRatio = 10/100;
                        net.divideParam.testRatio = 10/100;

                        %PARÁMETROS DE ENTRENAMIENTO
                        net.trainParam.epochs = 1000; %Número
máximo de iteraciones
                        net.trainParam.goal = 0; %Error objetivo
                        net.trainParam.time = inf;%tiempo máximo de
entrenamiento
                        net.trainParam.min_grad = 1e-5;%Gradiente
mínimo
                        net.trainParam.lr = 0.06;%Mu tasa de
aprendizaje
                        net.trainParam.max_fail=30;% Iteraciones
despues de encontrar un mínimo

                        net = initlay(net);%inicialización de pesos
```

```
                        % ENTRENAMIENTO DE LA RED
                        %%%%%
                        [net,tr] = train(net,inputsH,target);

                        %GeneraciOn de pesos
                        %%%%
                        PesoEntradasF= net.IW;
                        PesosCapasF = net.LW;
                        PesosBiasF=net.b; %representa la constante


PesosCapas1=abs(cell2mat(PesoEntradasF(1,1)));

Pesos_Entrada(a,b)=struct('pesos',(cell2mat(PesoEntradasF(1,1))));

                        PesosCapas2=abs(cell2mat(PesosCapasF(2,1)));

Pesos_Salida(a,b)=struct('pesos',(cell2mat(PesosCapasF(2,1))));


PesosBias(a,b)=struct('pesos',(cell2mat(PesosBiasF(1,1))));


                        %Q(im) peso de la variable i a la salida m,
                        %Guardado en un arreglo matricial por
iteración

                         for I=1:1:k
                         num1=0;%numerador de la ecuaciOn en 0 para
acumular y generar la sumatoria
                         den2=0;%denominador de la ecuaciOn en 0
para acumular y generar la sumatoria
                             for i=1:1:k
                                  for j=1:1:l
                                       O=sum(PesosCapas1');  %O es
un vector sumatoria r,j

num1=(PesosCapas1(j,I)./O(j)).*PesosCapas2(j)+num1;

den2=(PesosCapas1(j,i)./O(j)).*PesosCapas2(j)+den2;
                                       Q(a,b,I)=num1./den2;
                                  end
                             end
                         end


                        format short %Banderas creadas para hacer
seguimiento de las iteraciones en caso de ser necesario

                        red(a,b,l) =
struct('cod',((10000)+(k*100)+(l)),'dir',tr);


                        %DEFINICIÓN DE VARIABLES PARA ANÁLISIS DE
CORRELACIÓN Y ERRORES
                        %%%%%%%%
                        output = net(inputsH);
```

```
                        trOut = output(tr.trainInd);
                        vOut = output(tr.valInd);
                        tsOut = output(tr.testInd);
                        trTarg = target(tr.trainInd);
                        vTarg = target(tr.valInd);
                        tsTarg = target(tr.testInd);

                         %Regresiones y R^2
                         mdl = LinearModel.fit(trTarg,trOut);
                         T=mdl.Rsquared.Adjusted;
                         R2T(a,b)=T;
                         mdl = LinearModel.fit(vTarg,vOut);
                         V=mdl.Rsquared.Adjusted;
                         R2V(a,b)=V;
                         mdl = LinearModel.fit(tsTarg,tsOut);
                         TS=mdl.Rsquared.Adjusted;
                         R2TS(a,b)=TS;

                        %analisis del error
                        mseT=mean((trOut-trTarg).^2);
                        mseV=mean((vTarg-vOut).^2);
                        mseTS=mean((tsTarg-tsOut).^2);

                        MSET(a,b)= mseT;
                        MSEV(a,b)= mseV;
                        MSETS(a,b)= mseTS;

                        Qdos(a,b) = 1-(mseV/mseT);

                        b=b+1;

                    end

          a=a+1;

               case 8
                  b=1;

                    for l= round(k/2):1:(k*2) % numero de neuronas
                        disp([(n),(b),(k)])
                       %%%%%%%%%%%%%
                       % CREAR RNN %
                       %%%%%%%%%%%%%
                       %ARQUITECTURA
                        net = newff(minmax(inputsH),[l 1],{'logsig'
'purelin'},'trainlm');
                        %[neuronas de entrada y salida],
                        %{funciones de activación logarítmica
sigmoidal y lineal}
                        % Algortimo de aprendizaje
                        net.performFcn = 'mse';% error cuadrático
medio

                       %DIVISIÓN DE DATPS

                        net.divideFcn = 'dividerand';  % División
aleatoria
                        net.divideMode = 'sample';  % Divide up
every sample
```

```
                        net.divideParam.trainRatio = 80/100;
                        net.divideParam.valRatio = 10/100;
                        net.divideParam.testRatio = 10/100;

                        %PARÁMETROS DE ENTRENAMIENTO
                        net.trainParam.epochs = 1000; %Número
máximo de iteraciones
                        net.trainParam.goal = 0; %Error objetivo
                        net.trainParam.time = inf;%tiempo máximo de
entrenamiento
                        net.trainParam.min_grad = 1e-5;%Gradiente
mínimo
                        net.trainParam.lr = 0.06;%Mu tasa de
aprendizaje
                        net.trainParam.max_fail=30;% Iteraciones
despues de encontrar un mínimo

                        net = initlay(net);%inicialización de pesos


                        % ENTRENAMIENTO DE LA RED
                       %%%%%
                       [net,tr] = train(net,inputsH,target);

                       %GeneraciOn de pesos
                       %%%%
                       PesoEntradasF= net.IW;
                       PesosCapasF = net.LW;
                       PesosBiasF=net.b; %representa la constante


PesosCapas1=abs(cell2mat(PesoEntradasF(1,1)));

Pesos_Entrada(a,b)=struct('pesos',(cell2mat(PesoEntradasF(1,1))));

                      PesosCapas2=abs(cell2mat(PesosCapasF(2,1)));

Pesos_Salida(a,b)=struct('pesos',(cell2mat(PesosCapasF(2,1))));


PesosBias(a,b)=struct('pesos',(cell2mat(PesosBiasF(1,1))));


                       %Q(im) peso de la variable i a la salida m,
                       %Guardado en un arreglo matricial por
iteración

                        for I=1:1:k
                        num1=0;%numerador de la ecuaciOn en 0 para
acumular y generar la sumatoria
                        den2=0;%denominador de la ecuaciOn en 0
para acumular y generar la sumatoria
                            for i=1:1:k
                                 for j=1:1:l
                                      O=sum(PesosCapas1');  %O es
un vector sumatoria r,j

num1=(PesosCapas1(j,I)./O(j)).*PesosCapas2(j)+num1;
```

```
den2=(PesosCapas1(j,i)./O(j)).*PesosCapas2(j)+den2;
                                        Q(a,b,I)=num1./den2;
                                    end
                                end
                            end


                            format short %Banderas creadas para hacer
seguimiento de las iteraciones en caso de ser necesario

                            red(a,b,l) =
struct('cod',((10000)+(k*100)+(l)),'dir',tr);


                            %DEFINICIÓN DE VARIABLES PARA ANÁLISIS DE
CORRELACIÓN Y ERRORES
                            %%%%%%%%
                            output = net(inputsH);
                            trOut = output(tr.trainInd);
                            vOut = output(tr.valInd);
                            tsOut = output(tr.testInd);
                            trTarg = target(tr.trainInd);
                            vTarg = target(tr.valInd);
                            tsTarg = target(tr.testInd);

                             %Regresiones y R^2
                             mdl = LinearModel.fit(trTarg,trOut);
                             T=mdl.Rsquared.Adjusted;
                             R2T(a,b)=T;
                             mdl = LinearModel.fit(vTarg,vOut);
                             V=mdl.Rsquared.Adjusted;
                             R2V(a,b)=V;
                             mdl = LinearModel.fit(tsTarg,tsOut);
                             TS=mdl.Rsquared.Adjusted;
                             R2TS(a,b)=TS;

                            %analisis del error
                            mseT=mean((trOut-trTarg).^2);
                            mseV=mean((vTarg-vOut).^2);
                            mseTS=mean((tsTarg-tsOut).^2);

                            MSET(a,b)= mseT;
                            MSEV(a,b)= mseV;
                            MSETS(a,b)= mseTS;

                            Qdos(a,b) = 1-(mseV/mseT);

                            b=b+1;

                        end

          a=a+1;

               case 9
                  b=1;

                    for l= round(k/2):1:(k*2) % numero de neuronas
                        disp([(n),(b),(k)])
```

```
                        %%%%%%%%%%%%%
                        % CREAR RNN %
                        %%%%%%%%%%%%%
                        %ARQUITECTURA
                         net = newff(minmax(inputsH),[l 1],{'logsig'
'purelin'},'trainlm');
                         %[neuronas de entrada y salida],
                         %{funciones de activación logarítmica
sigmoidal y lineal}
                         % Algortimo de aprendizaje
                         net.performFcn = 'mse';% error cuadrático
medio

                        %DIVISIÓN DE DATPS

                         net.divideFcn = 'dividerand';  % División
aleatoria
                         net.divideMode = 'sample';  % Divide up
every sample
                         net.divideParam.trainRatio = 80/100;
                         net.divideParam.valRatio = 10/100;
                         net.divideParam.testRatio = 10/100;

                         %PARÁMETROS DE ENTRENAMIENTO
                         net.trainParam.epochs = 1000; %Número
máximo de iteraciones
                         net.trainParam.goal = 0; %Error objetivo
                         net.trainParam.time = inf;%tiempo máximo de
entrenamiento
                         net.trainParam.min_grad = 1e-5;%Gradiente
mínimo
                         net.trainParam.lr = 0.06;%Mu tasa de
aprendizaje
                         net.trainParam.max_fail=30;% Iteraciones
despues de encontrar un mínimo

                         net = initlay(net);%inicialización de pesos


                         % ENTRENAMIENTO DE LA RED
                        %%%%%
                        [net,tr] = train(net,inputsH,target);

                        %GeneraciOn de pesos
                        %%%%
                        PesoEntradasF= net.IW;
                        PesosCapasF = net.LW;
                        PesosBiasF=net.b; %representa la constante


PesosCapas1=abs(cell2mat(PesoEntradasF(1,1)));

Pesos_Entrada(a,b)=struct('pesos',(cell2mat(PesoEntradasF(1,1))));

                        PesosCapas2=abs(cell2mat(PesosCapasF(2,1)));

Pesos_Salida(a,b)=struct('pesos',(cell2mat(PesosCapasF(2,1))));
```

```
PesosBias(a,b)=struct('pesos',(cell2mat(PesosBiasF(1,1))));


                        %Q(im) peso de la variable i a la salida m,
                        %Guardado en un arreglo matricial por
iteración

                         for I=1:1:k
                         num1=0;%numerador de la ecuaciOn en 0 para
acumular y generar la sumatoria
                         den2=0;%denominador de la ecuaciOn en 0
para acumular y generar la sumatoria
                             for i=1:1:k
                                  for j=1:1:l
                                       O=sum(PesosCapas1');  %O es
un vector sumatoria r,j

num1=(PesosCapas1(j,I)./O(j)).*PesosCapas2(j)+num1;

den2=(PesosCapas1(j,i)./O(j)).*PesosCapas2(j)+den2;
                                       Q(a,b,I)=num1./den2;
                                  end
                             end
                         end


                         format short %Banderas creadas para hacer
seguimiento de las iteraciones en caso de ser necesario

                         red(a,b,l) =
struct('cod',((10000)+(k*100)+(l)),'dir',tr);


                         %DEFINICIÓN DE VARIABLES PARA ANÁLISIS DE
CORRELACIÓN Y ERRORES
                         %%%%%%%%
                         output = net(inputsH);
                         trOut = output(tr.trainInd);
                         vOut = output(tr.valInd);
                         tsOut = output(tr.testInd);
                         trTarg = target(tr.trainInd);
                         vTarg = target(tr.valInd);
                         tsTarg = target(tr.testInd);

                          %Regresiones y R^2
                          mdl = LinearModel.fit(trTarg,trOut);
                          T=mdl.Rsquared.Adjusted;
                          R2T(a,b)=T;
                          mdl = LinearModel.fit(vTarg,vOut);
                          V=mdl.Rsquared.Adjusted;
                          R2V(a,b)=V;
                          mdl = LinearModel.fit(tsTarg,tsOut);
                          TS=mdl.Rsquared.Adjusted;
                          R2TS(a,b)=TS;

                         %analisis del error
                         mseT=mean((trOut-trTarg).^2);
                         mseV=mean((vTarg-vOut).^2);
```

```
                        mseTS=mean((tsTarg-tsOut).^2);

                        MSET(a,b)= mseT;
                        MSEV(a,b)= mseV;
                        MSETS(a,b)= mseTS;

                        Qdos(a,b) = 1-(mseV/mseT);

                        b=b+1;

                    end

          a=a+1;

              case 10
                  b=1;

                    for l= round(k/2):1:(k*2) % numero de neuronas
                        disp([(n),(b),(k)])
                       %%%%%%%%%%%%%
                       % CREAR RNN %
                       %%%%%%%%%%%%%
                       %ARQUITECTURA
                        net = newff(minmax(inputsH),[l 1],{'logsig'
'purelin'},'trainlm');
                        %[neuronas de entrada y salida],
                        %{funciones de activación logarítmica
sigmoidal y lineal}
                        % Algortimo de aprendizaje
                        net.performFcn = 'mse';% error cuadrático
medio

                       %DIVISIÓN DE DATPS

                        net.divideFcn = 'dividerand';  % División
aleatoria
                        net.divideMode = 'sample';  % Divide up
every sample
                        net.divideParam.trainRatio = 80/100;
                        net.divideParam.valRatio = 10/100;
                        net.divideParam.testRatio = 10/100;

                        %PARÁMETROS DE ENTRENAMIENTO
                        net.trainParam.epochs = 1000; %Número
máximo de iteraciones
                        net.trainParam.goal = 0; %Error objetivo
                        net.trainParam.time = inf;%tiempo máximo de
entrenamiento
                        net.trainParam.min_grad = 1e-5;%Gradiente
mínimo
                        net.trainParam.lr = 0.06;%Mu tasa de
aprendizaje
                        net.trainParam.max_fail=30;% Iteraciones
despues de encontrar un mínimo

                        net = initlay(net);%inicialización de pesos


                        % ENTRENAMIENTO DE LA RED
```

```
                        %%%%%
                        [net,tr] = train(net,inputsH,target);

                        %GeneraciOn de pesos
                        %%%%
                        PesoEntradasF= net.IW;
                        PesosCapasF = net.LW;
                        PesosBiasF=net.b; %representa la constante


PesosCapas1=abs(cell2mat(PesoEntradasF(1,1)));

Pesos_Entrada(a,b)=struct('pesos',(cell2mat(PesoEntradasF(1,1))));

                        PesosCapas2=abs(cell2mat(PesosCapasF(2,1)));

Pesos_Salida(a,b)=struct('pesos',(cell2mat(PesosCapasF(2,1))));


PesosBias(a,b)=struct('pesos',(cell2mat(PesosBiasF(1,1))));


                        %Q(im) peso de la variable i a la salida m,
                        %Guardado en un arreglo matricial por
iteración

                         for I=1:1:k
                         num1=0;%numerador de la ecuaciOn en 0 para
acumular y generar la sumatoria
                         den2=0;%denominador de la ecuaciOn en 0
para acumular y generar la sumatoria
                             for i=1:1:k
                                  for j=1:1:l
                                       O=sum(PesosCapas1');  %O es
un vector sumatoria r,j

num1=(PesosCapas1(j,I)./O(j)).*PesosCapas2(j)+num1;

den2=(PesosCapas1(j,i)./O(j)).*PesosCapas2(j)+den2;
                                       Q(a,b,I)=num1./den2;
                                  end
                             end
                         end


                         format short %Banderas creadas para hacer
seguimiento de las iteraciones en caso de ser necesario

                         red(a,b,l) =
struct('cod',((10000)+(k*100)+(l)),'dir',tr);


                         %DEFINICIÓN DE VARIABLES PARA ANÁLISIS DE
CORRELACIÓN Y ERRORES
                         %%%%%%%%
                         output = net(inputsH);
                         trOut = output(tr.trainInd);
                         vOut = output(tr.valInd);
                         tsOut = output(tr.testInd);
```

```
                        trTarg = target(tr.trainInd);
                        vTarg = target(tr.valInd);
                        tsTarg = target(tr.testInd);

                         %Regresiones y R^2
                         mdl = LinearModel.fit(trTarg,trOut);
                         T=mdl.Rsquared.Adjusted;
                         R2T(a,b)=T;
                         mdl = LinearModel.fit(vTarg,vOut);
                         V=mdl.Rsquared.Adjusted;
                         R2V(a,b)=V;
                         mdl = LinearModel.fit(tsTarg,tsOut);
                         TS=mdl.Rsquared.Adjusted;
                         R2TS(a,b)=TS;

                        %analisis del error
                        mseT=mean((trOut-trTarg).^2);
                        mseV=mean((vTarg-vOut).^2);
                        mseTS=mean((tsTarg-tsOut).^2);

                        MSET(a,b)= mseT;
                        MSEV(a,b)= mseV;
                        MSETS(a,b)= mseTS;

                        Qdos(a,b) = 1-(mseV/mseT);

                        b=b+1;

                    end

          a=a+1;

        end
    end

end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%% VIENTO %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%función de anidados para generar 10 grupos de iteraciones

function
[red,Qdos,Q,R2T,R2V,R2TS,MSEV,MSET]=VIENTO(V_In11,V_target)

    target = V_target.mat;
    inputsV = V_In11.mat;
    k=size(inputsV);
    k=k(1);% numero de variables
    a=1;
    for n = 1:10
        switch n
```

```
            case 1
                 b=1;

                    for l= round(k/2):1:(k*2) % numero de neuronas
                        disp([(n),(b),(k)])
                       %%%%%%%%%%%%%
                       % CREAR RNN %
                       %%%%%%%%%%%%%
                       %ARQUITECTURA
                        net = newff(minmax(inputsV),[l 1],{'logsig'
'purelin'},'trainlm');
                        %[neuronas de entrada y salida],
                        %{funciones de activación logarítmica
sigmoidal y lineal}
                        % Algortimo de aprendizaje
                        net.performFcn = 'mse';% error cuadrático
medio

                       %DIVISIÓN DE DATPS

                        net.divideFcn = 'dividerand';  % División
aleatoria
                        net.divideMode = 'sample';  % Divide up
every sample
                        net.divideParam.trainRatio = 80/100;
                        net.divideParam.valRatio = 10/100;
                        net.divideParam.testRatio = 10/100;

                        %PARÁMETROS DE ENTRENAMIENTO
                        net.trainParam.epochs = 1000; %Número
máximo de iteraciones
                        net.trainParam.goal = 0; %Error objetivo
                        net.trainParam.time = inf;%tiempo máximo de
entrenamiento
                        net.trainParam.min_grad = 1e-5;%Gradiente
mínimo
                        net.trainParam.lr = 0.06;%Mu tasa de
aprendizaje
                        net.trainParam.max_fail=30;% Iteraciones
despues de encontrar un mínimo

                        net = initlay(net);%inicialización de pesos


                        % ENTRENAMIENTO DE LA RED
                       %%%%%
                       [net,tr] = train(net,inputsV,target);

                       %GeneraciOn de pesos
                       %%%%
                       PesoEntradasF= net.IW;
                       PesosCapasF = net.LW;
                       PesosBiasF=net.b; %representa la constante


PesosCapas1=abs(cell2mat(PesoEntradasF(1,1)));

Pesos_Entrada(a,b)=struct('pesos',(cell2mat(PesoEntradasF(1,1))));
```

```
                        PesosCapas2=abs(cell2mat(PesosCapasF(2,1)));

Pesos_Salida(a,b)=struct('pesos',(cell2mat(PesosCapasF(2,1))));


PesosBias(a,b)=struct('pesos',(cell2mat(PesosBiasF(1,1))));


                       %Q(im) peso de la variable i a la salida m,
                       %Guardado en un arreglo matricial por
iteración

                        for I=1:1:k
                        num1=0;%numerador de la ecuaciOn en 0 para
acumular y generar la sumatoria
                        den2=0;%denominador de la ecuaciOn en 0
para acumular y generar la sumatoria
                            for i=1:1:k
                                 for j=1:1:l
                                      O=sum(PesosCapas1');  %O es
un vector sumatoria r,j

num1=(PesosCapas1(j,I)./O(j)).*PesosCapas2(j)+num1;

den2=(PesosCapas1(j,i)./O(j)).*PesosCapas2(j)+den2;
                                      Q(a,b,I)=num1./den2;
                                 end
                            end
                        end


                        format short %Banderas creadas para hacer
seguimiento de las iteraciones en caso de ser necesario

                        red(a,b,l) =
struct('cod',((10000)+(k*100)+(l)),'dir',tr);


                        %DEFINICIÓN DE VARIABLES PARA ANÁLISIS DE
CORRELACIÓN Y ERRORES
                        %%%%%%%%
                        output = net(inputsV);
                        trOut = output(tr.trainInd);
                        vOut = output(tr.valInd);
                        tsOut = output(tr.testInd);
                        trTarg = target(tr.trainInd);
                        vTarg = target(tr.valInd);
                        tsTarg = target(tr.testInd);

                         %Regresiones y R^2
                         mdl = LinearModel.fit(trTarg,trOut);
                         T=mdl.Rsquared.Adjusted;
                         R2T(a,b)=T;
                         mdl = LinearModel.fit(vTarg,vOut);
                         V=mdl.Rsquared.Adjusted;
                         R2V(a,b)=V;
                         mdl = LinearModel.fit(tsTarg,tsOut);
                         TS=mdl.Rsquared.Adjusted;
                         R2TS(a,b)=TS;
```

```
                        %analisis del error
                        mseT=mean((trOut-trTarg).^2);
                        mseV=mean((vTarg-vOut).^2);
                        mseTS=mean((tsTarg-tsOut).^2);

                        MSET(a,b)= mseT;
                        MSEV(a,b)= mseV;
                        MSETS(a,b)= mseTS;

                        Qdos(a,b) = 1-(mseV/mseT);

                        b=b+1;

                    end

        a=a+1;

        case 2
        disp((n))

                b=1;

                  for l= round(k/2):1:(k*2) % numero de neuronas
                      disp([(n),(b),(k)])
                     %%%%%%%%%%%%%
                     % CREAR RNN %
                     %%%%%%%%%%%%%
                     %ARQUITECTURA
                      net = newff(minmax(inputsV),[l 1],{'logsig'
'purelin'},'trainlm');
                      %[neuronas de entrada y salida],
                      %{funciones de activación logarítmica
sigmoidal y lineal}
                      % Algortimo de aprendizaje
                      net.performFcn = 'mse';% error cuadrático
medio

                     %DIVISIÓN DE DATPS

                      net.divideFcn = 'dividerand';  % División
aleatoria
                      net.divideMode = 'sample';  % Divide up
every sample
                      net.divideParam.trainRatio = 80/100;
                      net.divideParam.valRatio = 10/100;
                      net.divideParam.testRatio = 10/100;

                      %PARÁMETROS DE ENTRENAMIENTO
                      net.trainParam.epochs = 1000; %Número
máximo de iteraciones
                      net.trainParam.goal = 0; %Error objetivo
                      net.trainParam.time = inf;%tiempo máximo de
entrenamiento
                      net.trainParam.min_grad = 1e-5;%Gradiente
mínimo
                      net.trainParam.lr = 0.06;%Mu tasa de
aprendizaje
```

```
                        net.trainParam.max_fail=30;% Iteraciones
despues de encontrar un mínimo

                        net = initlay(net);%inicialización de pesos


                        % ENTRENAMIENTO DE LA RED
                       %%%%%
                       [net,tr] = train(net,inputsV,target);

                       %GeneraciOn de pesos
                       %%%%
                       PesoEntradasF= net.IW;
                       PesosCapasF = net.LW;
                       PesosBiasF=net.b; %representa la constante


PesosCapas1=abs(cell2mat(PesoEntradasF(1,1)));

Pesos_Entrada(a,b)=struct('pesos',(cell2mat(PesoEntradasF(1,1))));

                       PesosCapas2=abs(cell2mat(PesosCapasF(2,1)));

Pesos_Salida(a,b)=struct('pesos',(cell2mat(PesosCapasF(2,1))));


PesosBias(a,b)=struct('pesos',(cell2mat(PesosBiasF(1,1))));


                       %Q(im) peso de la variable i a la salida m,
                       %Guardado en un arreglo matricial por
iteración

                        for I=1:1:k
                        num1=0;%numerador de la ecuaciOn en 0 para
acumular y generar la sumatoria
                        den2=0;%denominador de la ecuaciOn en 0
para acumular y generar la sumatoria
                            for i=1:1:k
                                 for j=1:1:l
                                      O=sum(PesosCapas1');  %O es
un vector sumatoria r,j

num1=(PesosCapas1(j,I)./O(j)).*PesosCapas2(j)+num1;

den2=(PesosCapas1(j,i)./O(j)).*PesosCapas2(j)+den2;
                                      Q(a,b,I)=num1./den2;
                                 end
                            end
                        end


                        format short %Banderas creadas para hacer
seguimiento de las iteraciones en caso de ser necesario

                        red(a,b,l) =
struct('cod',((10000)+(k*100)+(l)),'dir',tr);
```

```
                        %DEFINICIÓN DE VARIABLES PARA ANÁLISIS DE
CORRELACIÓN Y ERRORES
                        %%%%%%%%
                        output = net(inputsV);
                        trOut = output(tr.trainInd);
                        vOut = output(tr.valInd);
                        tsOut = output(tr.testInd);
                        trTarg = target(tr.trainInd);
                        vTarg = target(tr.valInd);
                        tsTarg = target(tr.testInd);

                         %Regresiones y R^2
                         mdl = LinearModel.fit(trTarg,trOut);
                         T=mdl.Rsquared.Adjusted;
                         R2T(a,b)=T;
                         mdl = LinearModel.fit(vTarg,vOut);
                         V=mdl.Rsquared.Adjusted;
                         R2V(a,b)=V;
                         mdl = LinearModel.fit(tsTarg,tsOut);
                         TS=mdl.Rsquared.Adjusted;
                         R2TS(a,b)=TS;

                        %analisis del error
                        mseT=mean((trOut-trTarg).^2);
                        mseV=mean((vTarg-vOut).^2);
                        mseTS=mean((tsTarg-tsOut).^2);

                        MSET(a,b)= mseT;
                        MSEV(a,b)= mseV;
                        MSETS(a,b)= mseTS;

                        Qdos(a,b) = 1-(mseV/mseT);

                        b=b+1;

                    end
                    a=a+1;

            case 3
                 b=1;

                    for l= round(k/2):1:(k*2) % numero de neuronas
                        disp([(n),(b),(k)])
                       %%%%%%%%%%%%%
                       % CREAR RNN %
                       %%%%%%%%%%%%%
                       %ARQUITECTURA
                        net = newff(minmax(inputsV),[l 1],{'logsig'
'purelin'},'trainlm');
                        %[neuronas de entrada y salida],
                        %{funciones de activación logarítmica
sigmoidal y lineal}
                        % Algortimo de aprendizaje
                        net.performFcn = 'mse';% error cuadrático
medio

                      %DIVISIÓN DE DATPS
```

```
                        net.divideFcn = 'dividerand';  % División
aleatoria
                        net.divideMode = 'sample';  % Divide up
every sample
                        net.divideParam.trainRatio = 80/100;
                        net.divideParam.valRatio = 10/100;
                        net.divideParam.testRatio = 10/100;

                        %PARÁMETROS DE ENTRENAMIENTO
                        net.trainParam.epochs = 1000; %Número
máximo de iteraciones
                        net.trainParam.goal = 0; %Error objetivo
                        net.trainParam.time = inf;%tiempo máximo de
entrenamiento
                        net.trainParam.min_grad = 1e-5;%Gradiente
mínimo
                        net.trainParam.lr = 0.06;%Mu tasa de
aprendizaje
                        net.trainParam.max_fail=30;% Iteraciones
despues de encontrar un mínimo

                        net = initlay(net);%inicialización de pesos


                         % ENTRENAMIENTO DE LA RED
                        %%%%%
                        [net,tr] = train(net,inputsV,target);

                        %GeneraciOn de pesos
                        %%%%
                        PesoEntradasF= net.IW;
                        PesosCapasF = net.LW;
                        PesosBiasF=net.b; %representa la constante


PesosCapas1=abs(cell2mat(PesoEntradasF(1,1)));

Pesos_Entrada(a,b)=struct('pesos',(cell2mat(PesoEntradasF(1,1))));

                       PesosCapas2=abs(cell2mat(PesosCapasF(2,1)));

Pesos_Salida(a,b)=struct('pesos',(cell2mat(PesosCapasF(2,1))));


PesosBias(a,b)=struct('pesos',(cell2mat(PesosBiasF(1,1))));


                        %Q(im) peso de la variable i a la salida m,
                        %Guardado en un arreglo matricial por
iteración

                         for I=1:1:k
                         num1=0;%numerador de la ecuaciOn en 0 para
acumular y generar la sumatoria
                         den2=0;%denominador de la ecuaciOn en 0
para acumular y generar la sumatoria
                             for i=1:1:k
                                  for j=1:1:l
```

```
                                        O=sum(PesosCapas1');  %O es
un vector sumatoria r,j

num1=(PesosCapas1(j,I)./O(j)).*PesosCapas2(j)+num1;

den2=(PesosCapas1(j,i)./O(j)).*PesosCapas2(j)+den2;
                                        Q(a,b,I)=num1./den2;
                                    end
                                end
                            end


                            format short %Banderas creadas para hacer
seguimiento de las iteraciones en caso de ser necesario

                            red(a,b,l) =
struct('cod',((10000)+(k*100)+(l)),'dir',tr);


                            %DEFINICIÓN DE VARIABLES PARA ANÁLISIS DE
CORRELACIÓN Y ERRORES
                            %%%%%%%%
                            output = net(inputsV);
                            trOut = output(tr.trainInd);
                            vOut = output(tr.valInd);
                            tsOut = output(tr.testInd);
                            trTarg = target(tr.trainInd);
                            vTarg = target(tr.valInd);
                            tsTarg = target(tr.testInd);

                             %Regresiones y R^2
                             mdl = LinearModel.fit(trTarg,trOut);
                             T=mdl.Rsquared.Adjusted;
                             R2T(a,b)=T;
                             mdl = LinearModel.fit(vTarg,vOut);
                             V=mdl.Rsquared.Adjusted;
                             R2V(a,b)=V;
                             mdl = LinearModel.fit(tsTarg,tsOut);
                             TS=mdl.Rsquared.Adjusted;
                             R2TS(a,b)=TS;

                            %analisis del error
                            mseT=mean((trOut-trTarg).^2);
                            mseV=mean((vTarg-vOut).^2);
                            mseTS=mean((tsTarg-tsOut).^2);

                            MSET(a,b)= mseT;
                            MSEV(a,b)= mseV;
                            MSETS(a,b)= mseTS;

                            Qdos(a,b) = 1-(mseV/mseT);

                            b=b+1;

                        end

                a=a+1;
```

```
            case 4
                 b=1;

                    for l= round(k/2):1:(k*2) % numero de neuronas
                        disp([(n),(b),(k)])
                       %%%%%%%%%%%%%
                       % CREAR RNN %
                       %%%%%%%%%%%%%
                       %ARQUITECTURA
                        net = newff(minmax(inputsV),[l 1],{'logsig'
'purelin'},'trainlm');
                        %[neuronas de entrada y salida],
                        %{funciones de activación logarítmica
sigmoidal y lineal}
                        % Algortimo de aprendizaje
                        net.performFcn = 'mse';% error cuadrático
medio

                       %DIVISIÓN DE DATPS

                        net.divideFcn = 'dividerand';  % División
aleatoria
                        net.divideMode = 'sample';  % Divide up
every sample
                        net.divideParam.trainRatio = 80/100;
                        net.divideParam.valRatio = 10/100;
                        net.divideParam.testRatio = 10/100;

                        %PARÁMETROS DE ENTRENAMIENTO
                        net.trainParam.epochs = 1000; %Número
máximo de iteraciones
                        net.trainParam.goal = 0; %Error objetivo
                        net.trainParam.time = inf;%tiempo máximo de
entrenamiento
                        net.trainParam.min_grad = 1e-5;%Gradiente
mínimo
                        net.trainParam.lr = 0.06;%Mu tasa de
aprendizaje
                        net.trainParam.max_fail=30;% Iteraciones
despues de encontrar un mínimo

                        net = initlay(net);%inicialización de pesos


                        % ENTRENAMIENTO DE LA RED
                       %%%%%
                       [net,tr] = train(net,inputsV,target);

                       %GeneraciOn de pesos
                       %%%%
                       PesoEntradasF= net.IW;
                       PesosCapasF = net.LW;
                       PesosBiasF=net.b; %representa la constante


PesosCapas1=abs(cell2mat(PesoEntradasF(1,1)));

Pesos_Entrada(a,b)=struct('pesos',(cell2mat(PesoEntradasF(1,1))));
```

```
                        PesosCapas2=abs(cell2mat(PesosCapasF(2,1)));

Pesos_Salida(a,b)=struct('pesos',(cell2mat(PesosCapasF(2,1))));


PesosBias(a,b)=struct('pesos',(cell2mat(PesosBiasF(1,1))));


                       %Q(im) peso de la variable i a la salida m,
                       %Guardado en un arreglo matricial por
iteración

                        for I=1:1:k
                        num1=0;%numerador de la ecuaciOn en 0 para
acumular y generar la sumatoria
                        den2=0;%denominador de la ecuaciOn en 0
para acumular y generar la sumatoria
                            for i=1:1:k
                                 for j=1:1:l
                                      O=sum(PesosCapas1');  %O es
un vector sumatoria r,j

num1=(PesosCapas1(j,I)./O(j)).*PesosCapas2(j)+num1;

den2=(PesosCapas1(j,i)./O(j)).*PesosCapas2(j)+den2;
                                      Q(a,b,I)=num1./den2;
                                 end
                            end
                        end


                        format short %Banderas creadas para hacer
seguimiento de las iteraciones en caso de ser necesario

                        red(a,b,l) =
struct('cod',((10000)+(k*100)+(l)),'dir',tr);


                        %DEFINICIÓN DE VARIABLES PARA ANÁLISIS DE
CORRELACIÓN Y ERRORES
                        %%%%%%%%
                        output = net(inputsV);
                        trOut = output(tr.trainInd);
                        vOut = output(tr.valInd);
                        tsOut = output(tr.testInd);
                        trTarg = target(tr.trainInd);
                        vTarg = target(tr.valInd);
                        tsTarg = target(tr.testInd);

                         %Regresiones y R^2
                         mdl = LinearModel.fit(trTarg,trOut);
                         T=mdl.Rsquared.Adjusted;
                         R2T(a,b)=T;
                         mdl = LinearModel.fit(vTarg,vOut);
                         V=mdl.Rsquared.Adjusted;
                         R2V(a,b)=V;
                         mdl = LinearModel.fit(tsTarg,tsOut);
                         TS=mdl.Rsquared.Adjusted;
                         R2TS(a,b)=TS;
```

```
                        %analisis del error
                        mseT=mean((trOut-trTarg).^2);
                        mseV=mean((vTarg-vOut).^2);
                        mseTS=mean((tsTarg-tsOut).^2);

                        MSET(a,b)= mseT;
                        MSEV(a,b)= mseV;
                        MSETS(a,b)= mseTS;

                        Qdos(a,b) = 1-(mseV/mseT);

                        b=b+1;

                    end

          a=a+1;

             case 5
                  b=1;

                    for l= round(k/2):1:(k*2) % numero de neuronas
                        disp([(n),(b),(k)])
                       %%%%%%%%%%%%%
                       % CREAR RNN %
                       %%%%%%%%%%%%%
                       %ARQUITECTURA
                        net = newff(minmax(inputsV),[l 1],{'logsig'
'purelin'},'trainlm');
                        %[neuronas de entrada y salida],
                        %{funciones de activación logarítmica
sigmoidal y lineal}
                        % Algortimo de aprendizaje
                        net.performFcn = 'mse';% error cuadrático
medio

                       %DIVISIÓN DE DATPS

                        net.divideFcn = 'dividerand';  % División
aleatoria
                        net.divideMode = 'sample';  % Divide up
every sample
                        net.divideParam.trainRatio = 80/100;
                        net.divideParam.valRatio = 10/100;
                        net.divideParam.testRatio = 10/100;

                        %PARÁMETROS DE ENTRENAMIENTO
                        net.trainParam.epochs = 1000; %Número
máximo de iteraciones
                        net.trainParam.goal = 0; %Error objetivo
                        net.trainParam.time = inf;%tiempo máximo de
entrenamiento
                        net.trainParam.min_grad = 1e-5;%Gradiente
mínimo
                        net.trainParam.lr = 0.06;%Mu tasa de
aprendizaje
                        net.trainParam.max_fail=30;% Iteraciones
despues de encontrar un mínimo
```

```
                        net = initlay(net);%inicialización de pesos


                        % ENTRENAMIENTO DE LA RED
                        %%%%%
                        [net,tr] = train(net,inputsV,target);

                        %GeneraciOn de pesos
                        %%%%
                        PesoEntradasF= net.IW;
                        PesosCapasF = net.LW;
                        PesosBiasF=net.b; %representa la constante


PesosCapas1=abs(cell2mat(PesoEntradasF(1,1)));

Pesos_Entrada(a,b)=struct('pesos',(cell2mat(PesoEntradasF(1,1))));

                        PesosCapas2=abs(cell2mat(PesosCapasF(2,1)));

Pesos_Salida(a,b)=struct('pesos',(cell2mat(PesosCapasF(2,1))));


PesosBias(a,b)=struct('pesos',(cell2mat(PesosBiasF(1,1))));


                        %Q(im) peso de la variable i a la salida m,
                        %Guardado en un arreglo matricial por
iteración

                         for I=1:1:k
                         num1=0;%numerador de la ecuaciOn en 0 para
acumular y generar la sumatoria
                         den2=0;%denominador de la ecuaciOn en 0
para acumular y generar la sumatoria
                             for i=1:1:k
                                  for j=1:1:l
                                       O=sum(PesosCapas1');  %O es
un vector sumatoria r,j

num1=(PesosCapas1(j,I)./O(j)).*PesosCapas2(j)+num1;

den2=(PesosCapas1(j,i)./O(j)).*PesosCapas2(j)+den2;
                                            Q(a,b,I)=num1./den2;
                                  end
                             end
                         end


                         format short %Banderas creadas para hacer
seguimiento de las iteraciones en caso de ser necesario

                         red(a,b,l) =
struct('cod',((10000)+(k*100)+(l)),'dir',tr);


                         %DEFINICIÓN DE VARIABLES PARA ANÁLISIS DE
CORRELACIÓN Y ERRORES
                         %%%%%%%%
```

```
                        output = net(inputsV);
                        trOut = output(tr.trainInd);
                        vOut = output(tr.valInd);
                        tsOut = output(tr.testInd);
                        trTarg = target(tr.trainInd);
                        vTarg = target(tr.valInd);
                        tsTarg = target(tr.testInd);

                         %Regresiones y R^2
                         mdl = LinearModel.fit(trTarg,trOut);
                         T=mdl.Rsquared.Adjusted;
                         R2T(a,b)=T;
                         mdl = LinearModel.fit(vTarg,vOut);
                         V=mdl.Rsquared.Adjusted;
                         R2V(a,b)=V;
                         mdl = LinearModel.fit(tsTarg,tsOut);
                         TS=mdl.Rsquared.Adjusted;
                         R2TS(a,b)=TS;

                        %analisis del error
                        mseT=mean((trOut-trTarg).^2);
                        mseV=mean((vTarg-vOut).^2);
                        mseTS=mean((tsTarg-tsOut).^2);

                        MSET(a,b)= mseT;
                        MSEV(a,b)= mseV;
                        MSETS(a,b)= mseTS;

                        Qdos(a,b) = 1-(mseV/mseT);

                        b=b+1;

                    end

          a=a+1;

              case 6
                  b=1;

                    for l= round(k/2):1:(k*2) % numero de neuronas
                        disp([(n),(b),(k)])
                       %%%%%%%%%%%%%
                       % CREAR RNN %
                       %%%%%%%%%%%%%
                       %ARQUITECTURA
                        net = newff(minmax(inputsV),[l 1],{'logsig'
'purelin'},'trainlm');
                        %[neuronas de entrada y salida],
                        %{funciones de activación logarítmica
sigmoidal y lineal}
                        % Algortimo de aprendizaje
                        net.performFcn = 'mse';% error cuadrático
medio

                       %DIVISIÓN DE DATPS

                        net.divideFcn = 'dividerand';  % División
aleatoria
```

```
                        net.divideMode = 'sample';  % Divide up
every sample
                        net.divideParam.trainRatio = 80/100;
                        net.divideParam.valRatio = 10/100;
                        net.divideParam.testRatio = 10/100;

                        %PARÁMETROS DE ENTRENAMIENTO
                        net.trainParam.epochs = 1000; %Número
máximo de iteraciones
                        net.trainParam.goal = 0; %Error objetivo
                        net.trainParam.time = inf;%tiempo máximo de
entrenamiento
                        net.trainParam.min_grad = 1e-5;%Gradiente
mínimo
                        net.trainParam.lr = 0.06;%Mu tasa de
aprendizaje
                        net.trainParam.max_fail=30;% Iteraciones
despues de encontrar un mínimo

                        net = initlay(net);%inicialización de pesos


                         % ENTRENAMIENTO DE LA RED
                        %%%%%
                        [net,tr] = train(net,inputsV,target);

                        %GeneraciOn de pesos
                        %%%%
                        PesoEntradasF= net.IW;
                        PesosCapasF = net.LW;
                        PesosBiasF=net.b; %representa la constante


PesosCapas1=abs(cell2mat(PesoEntradasF(1,1)));

Pesos_Entrada(a,b)=struct('pesos',(cell2mat(PesoEntradasF(1,1))));

                       PesosCapas2=abs(cell2mat(PesosCapasF(2,1)));

Pesos_Salida(a,b)=struct('pesos',(cell2mat(PesosCapasF(2,1))));


PesosBias(a,b)=struct('pesos',(cell2mat(PesosBiasF(1,1))));


                        %Q(im) peso de la variable i a la salida m,
                        %Guardado en un arreglo matricial por
iteración

                         for I=1:1:k
                         num1=0;%numerador de la ecuaciOn en 0 para
acumular y generar la sumatoria
                          den2=0;%denominador de la ecuaciOn en 0
para acumular y generar la sumatoria
                              for i=1:1:k
                                   for j=1:1:l
                                        O=sum(PesosCapas1');  %O es
un vector sumatoria r,j
```

```
num1=(PesosCapas1(j,I)./O(j)).*PesosCapas2(j)+num1;

den2=(PesosCapas1(j,i)./O(j)).*PesosCapas2(j)+den2;
                                        Q(a,b,I)=num1./den2;
                                    end
                                end
                            end


                            format short %Banderas creadas para hacer
seguimiento de las iteraciones en caso de ser necesario

                            red(a,b,l) =
struct('cod',((10000)+(k*100)+(l)),'dir',tr);


                            %DEFINICIÓN DE VARIABLES PARA ANÁLISIS DE
CORRELACIÓN Y ERRORES
                            %%%%%%%%
                            output = net(inputsV);
                            trOut = output(tr.trainInd);
                            vOut = output(tr.valInd);
                            tsOut = output(tr.testInd);
                            trTarg = target(tr.trainInd);
                            vTarg = target(tr.valInd);
                            tsTarg = target(tr.testInd);

                             %Regresiones y R^2
                             mdl = LinearModel.fit(trTarg,trOut);
                             T=mdl.Rsquared.Adjusted;
                             R2T(a,b)=T;
                             mdl = LinearModel.fit(vTarg,vOut);
                             V=mdl.Rsquared.Adjusted;
                             R2V(a,b)=V;
                             mdl = LinearModel.fit(tsTarg,tsOut);
                             TS=mdl.Rsquared.Adjusted;
                             R2TS(a,b)=TS;

                            %analisis del error
                            mseT=mean((trOut-trTarg).^2);
                            mseV=mean((vTarg-vOut).^2);
                            mseTS=mean((tsTarg-tsOut).^2);

                            MSET(a,b)= mseT;
                            MSEV(a,b)= mseV;
                            MSETS(a,b)= mseTS;

                            Qdos(a,b) = 1-(mseV/mseT);

                            b=b+1;

                        end

                a=a+1;

                     case 7
                        b=1;
```

```
                    for l= round(k/2):1:(k*2) % numero de neuronas
                        disp([(n),(b),(k)])
                       %%%%%%%%%%%%%
                       % CREAR RNN %
                       %%%%%%%%%%%%%
                       %ARQUITECTURA
                        net = newff(minmax(inputsV),[l 1],{'logsig'
'purelin'},'trainlm');
                        %[neuronas de entrada y salida],
                        %{funciones de activación logarítmica
sigmoidal y lineal}
                        % Algortimo de aprendizaje
                        net.performFcn = 'mse';% error cuadrático
medio

                       %DIVISIÓN DE DATPS

                        net.divideFcn = 'dividerand';  % División
aleatoria
                        net.divideMode = 'sample';  % Divide up
every sample
                        net.divideParam.trainRatio = 80/100;
                        net.divideParam.valRatio = 10/100;
                        net.divideParam.testRatio = 10/100;

                        %PARÁMETROS DE ENTRENAMIENTO
                        net.trainParam.epochs = 1000; %Número
máximo de iteraciones
                        net.trainParam.goal = 0; %Error objetivo
                        net.trainParam.time = inf;%tiempo máximo de
entrenamiento
                        net.trainParam.min_grad = 1e-5;%Gradiente
mínimo
                        net.trainParam.lr = 0.06;%Mu tasa de
aprendizaje
                        net.trainParam.max_fail=30;% Iteraciones
despues de encontrar un mínimo

                        net = initlay(net);%inicialización de pesos


                        % ENTRENAMIENTO DE LA RED
                       %%%%%
                       [net,tr] = train(net,inputsV,target);

                       %GeneraciOn de pesos
                       %%%%
                       PesoEntradasF= net.IW;
                       PesosCapasF = net.LW;
                       PesosBiasF=net.b; %representa la constante


PesosCapas1=abs(cell2mat(PesoEntradasF(1,1)));

Pesos_Entrada(a,b)=struct('pesos',(cell2mat(PesoEntradasF(1,1))));

                       PesosCapas2=abs(cell2mat(PesosCapasF(2,1)));

Pesos_Salida(a,b)=struct('pesos',(cell2mat(PesosCapasF(2,1))));
```

```
PesosBias(a,b)=struct('pesos',(cell2mat(PesosBiasF(1,1))));


                        %Q(im) peso de la variable i a la salida m,
                        %Guardado en un arreglo matricial por
iteración

                         for I=1:1:k
                         num1=0;%numerador de la ecuaciOn en 0 para
acumular y generar la sumatoria
                         den2=0;%denominador de la ecuaciOn en 0
para acumular y generar la sumatoria
                             for i=1:1:k
                                  for j=1:1:l
                                       O=sum(PesosCapas1');  %O es
un vector sumatoria r,j

num1=(PesosCapas1(j,I)./O(j)).*PesosCapas2(j)+num1;

den2=(PesosCapas1(j,i)./O(j)).*PesosCapas2(j)+den2;
                                       Q(a,b,I)=num1./den2;
                                  end
                             end
                         end


                         format short %Banderas creadas para hacer
seguimiento de las iteraciones en caso de ser necesario

                         red(a,b,l) =
struct('cod',((10000)+(k*100)+(l)),'dir',tr);


                         %DEFINICIÓN DE VARIABLES PARA ANÁLISIS DE
CORRELACIÓN Y ERRORES
                         %%%%%%%%
                         output = net(inputsV);
                         trOut = output(tr.trainInd);
                         vOut = output(tr.valInd);
                         tsOut = output(tr.testInd);
                         trTarg = target(tr.trainInd);
                         vTarg = target(tr.valInd);
                         tsTarg = target(tr.testInd);

                          %Regresiones y R^2
                          mdl = LinearModel.fit(trTarg,trOut);
                          T=mdl.Rsquared.Adjusted;
                          R2T(a,b)=T;
                          mdl = LinearModel.fit(vTarg,vOut);
                          V=mdl.Rsquared.Adjusted;
                          R2V(a,b)=V;
                          mdl = LinearModel.fit(tsTarg,tsOut);
                          TS=mdl.Rsquared.Adjusted;
                          R2TS(a,b)=TS;

                         %analisis del error
                         mseT=mean((trOut-trTarg).^2);
```

```
                        mseV=mean((vTarg-vOut).^2);
                        mseTS=mean((tsTarg-tsOut).^2);

                        MSET(a,b)= mseT;
                        MSEV(a,b)= mseV;
                        MSETS(a,b)= mseTS;

                        Qdos(a,b) = 1-(mseV/mseT);

                        b=b+1;

                    end

          a=a+1;

               case 8
                  b=1;

                    for l= round(k/2):1:(k*2) % numero de neuronas
                        disp([(n),(b),(k)])
                       %%%%%%%%%%%%%
                       % CREAR RNN %
                       %%%%%%%%%%%%%
                       %ARQUITECTURA
                        net = newff(minmax(inputsV),[l 1],{'logsig'
'purelin'},'trainlm');
                        %[neuronas de entrada y salida],
                        %{funciones de activación logarítmica
sigmoidal y lineal}
                        % Algortimo de aprendizaje
                        net.performFcn = 'mse';% error cuadrático
medio

                       %DIVISIÓN DE DATPS

                        net.divideFcn = 'dividerand';  % División
aleatoria
                        net.divideMode = 'sample';  % Divide up
every sample
                        net.divideParam.trainRatio = 80/100;
                        net.divideParam.valRatio = 10/100;
                        net.divideParam.testRatio = 10/100;

                        %PARÁMETROS DE ENTRENAMIENTO
                        net.trainParam.epochs = 1000; %Número
máximo de iteraciones
                        net.trainParam.goal = 0; %Error objetivo
                        net.trainParam.time = inf;%tiempo máximo de
entrenamiento
                        net.trainParam.min_grad = 1e-5;%Gradiente
mínimo
                        net.trainParam.lr = 0.06;%Mu tasa de
aprendizaje
                        net.trainParam.max_fail=30;% Iteraciones
despues de encontrar un mínimo

                        net = initlay(net);%inicialización de pesos
```

```
                        % ENTRENAMIENTO DE LA RED
                        %%%%%
                        [net,tr] = train(net,inputsV,target);

                        %GeneraciOn de pesos
                        %%%%
                        PesoEntradasF= net.IW;
                        PesosCapasF = net.LW;
                        PesosBiasF=net.b; %representa la constante


PesosCapas1=abs(cell2mat(PesoEntradasF(1,1)));

Pesos_Entrada(a,b)=struct('pesos',(cell2mat(PesoEntradasF(1,1))));

                        PesosCapas2=abs(cell2mat(PesosCapasF(2,1)));

Pesos_Salida(a,b)=struct('pesos',(cell2mat(PesosCapasF(2,1))));


PesosBias(a,b)=struct('pesos',(cell2mat(PesosBiasF(1,1))));


                        %Q(im) peso de la variable i a la salida m,
                        %Guardado en un arreglo matricial por
iteración

                         for I=1:1:k
                         num1=0;%numerador de la ecuaciOn en 0 para
acumular y generar la sumatoria
                         den2=0;%denominador de la ecuaciOn en 0
para acumular y generar la sumatoria
                             for i=1:1:k
                                  for j=1:1:l
                                       O=sum(PesosCapas1');  %O es
un vector sumatoria r,j

num1=(PesosCapas1(j,I)./O(j)).*PesosCapas2(j)+num1;

den2=(PesosCapas1(j,i)./O(j)).*PesosCapas2(j)+den2;
                                       Q(a,b,I)=num1./den2;
                                  end
                             end
                         end


                         format short %Banderas creadas para hacer
seguimiento de las iteraciones en caso de ser necesario

                         red(a,b,l) =
struct('cod',((10000)+(k*100)+(l)),'dir',tr);


                         %DEFINICIÓN DE VARIABLES PARA ANÁLISIS DE
CORRELACIÓN Y ERRORES
                         %%%%%%%%
                         output = net(inputsV);
                         trOut = output(tr.trainInd);
                         vOut = output(tr.valInd);
```

```
                        tsOut = output(tr.testInd);
                        trTarg = target(tr.trainInd);
                        vTarg = target(tr.valInd);
                        tsTarg = target(tr.testInd);

                         %Regresiones y R^2
                         mdl = LinearModel.fit(trTarg,trOut);
                         T=mdl.Rsquared.Adjusted;
                         R2T(a,b)=T;
                         mdl = LinearModel.fit(vTarg,vOut);
                         V=mdl.Rsquared.Adjusted;
                         R2V(a,b)=V;
                         mdl = LinearModel.fit(tsTarg,tsOut);
                         TS=mdl.Rsquared.Adjusted;
                         R2TS(a,b)=TS;

                        %analisis del error
                        mseT=mean((trOut-trTarg).^2);
                        mseV=mean((vTarg-vOut).^2);
                        mseTS=mean((tsTarg-tsOut).^2);

                        MSET(a,b)= mseT;
                        MSEV(a,b)= mseV;
                        MSETS(a,b)= mseTS;

                        Qdos(a,b) = 1-(mseV/mseT);

                        b=b+1;

                    end

          a=a+1;

               case 9
                  b=1;

                    for l= round(k/2):1:(k*2) % numero de neuronas
                        disp([(n),(b),(k)])
                       %%%%%%%%%%%%%
                       % CREAR RNN %
                       %%%%%%%%%%%%%
                       %ARQUITECTURA
                        net = newff(minmax(inputsV),[l 1],{'logsig'
'purelin'},'trainlm');
                        %[neuronas de entrada y salida],
                        %{funciones de activación logarítmica
sigmoidal y lineal}
                        % Algortimo de aprendizaje
                        net.performFcn = 'mse';% error cuadrático
medio

                       %DIVISIÓN DE DATPS

                        net.divideFcn = 'dividerand';  % División
aleatoria
                        net.divideMode = 'sample';  % Divide up
every sample
                        net.divideParam.trainRatio = 80/100;
                        net.divideParam.valRatio = 10/100;
```

```
                        net.divideParam.testRatio = 10/100;

                        %PARÁMETROS DE ENTRENAMIENTO
                        net.trainParam.epochs = 1000; %Número
máximo de iteraciones
                        net.trainParam.goal = 0; %Error objetivo
                        net.trainParam.time = inf;%tiempo máximo de
entrenamiento
                        net.trainParam.min_grad = 1e-5;%Gradiente
mínimo
                        net.trainParam.lr = 0.06;%Mu tasa de
aprendizaje
                        net.trainParam.max_fail=30;% Iteraciones
despues de encontrar un mínimo

                        net = initlay(net);%inicialización de pesos


                        % ENTRENAMIENTO DE LA RED
                       %%%%%
                       [net,tr] = train(net,inputsV,target);

                       %GeneraciOn de pesos
                       %%%%
                       PesoEntradasF= net.IW;
                       PesosCapasF = net.LW;
                       PesosBiasF=net.b; %representa la constante


PesosCapas1=abs(cell2mat(PesoEntradasF(1,1)));

Pesos_Entrada(a,b)=struct('pesos',(cell2mat(PesoEntradasF(1,1))));

                      PesosCapas2=abs(cell2mat(PesosCapasF(2,1)));

Pesos_Salida(a,b)=struct('pesos',(cell2mat(PesosCapasF(2,1))));


PesosBias(a,b)=struct('pesos',(cell2mat(PesosBiasF(1,1))));


                       %Q(im) peso de la variable i a la salida m,
                       %Guardado en un arreglo matricial por
iteración

                        for I=1:1:k
                        num1=0;%numerador de la ecuaciOn en 0 para
acumular y generar la sumatoria
                        den2=0;%denominador de la ecuaciOn en 0
para acumular y generar la sumatoria
                            for i=1:1:k
                                 for j=1:1:l
                                      O=sum(PesosCapas1');  %O es
un vector sumatoria r,j

num1=(PesosCapas1(j,I)./O(j)).*PesosCapas2(j)+num1;

den2=(PesosCapas1(j,i)./O(j)).*PesosCapas2(j)+den2;
                                      Q(a,b,I)=num1./den2;
```

```
                        end
                    end
                end


                format short %Banderas creadas para hacer
seguimiento de las iteraciones en caso de ser necesario

                red(a,b,l) =
struct('cod',((10000)+(k*100)+(l)),'dir',tr);


                %DEFINICIÓN DE VARIABLES PARA ANÁLISIS DE
CORRELACIÓN Y ERRORES
                %%%%%%%%
                output = net(inputsV);
                trOut = output(tr.trainInd);
                vOut = output(tr.valInd);
                tsOut = output(tr.testInd);
                trTarg = target(tr.trainInd);
                vTarg = target(tr.valInd);
                tsTarg = target(tr.testInd);

                 %Regresiones y R^2
                 mdl = LinearModel.fit(trTarg,trOut);
                 T=mdl.Rsquared.Adjusted;
                 R2T(a,b)=T;
                 mdl = LinearModel.fit(vTarg,vOut);
                 V=mdl.Rsquared.Adjusted;
                 R2V(a,b)=V;
                 mdl = LinearModel.fit(tsTarg,tsOut);
                 TS=mdl.Rsquared.Adjusted;
                 R2TS(a,b)=TS;

                %analisis del error
                mseT=mean((trOut-trTarg).^2);
                mseV=mean((vTarg-vOut).^2);
                mseTS=mean((tsTarg-tsOut).^2);

                MSET(a,b)= mseT;
                MSEV(a,b)= mseV;
                MSETS(a,b)= mseTS;

                Qdos(a,b) = 1-(mseV/mseT);

                b=b+1;

            end

        a=a+1;

            case 10
                b=1;

                for l= round(k/2):1:(k*2) % numero de neuronas
                    disp([(n),(b),(k)])
                   %%%%%%%%%%%%%
                   % CREAR RNN %
                   %%%%%%%%%%%%%
```

```
                        %ARQUITECTURA
                         net = newff(minmax(inputsV),[l 1],{'logsig'
'purelin'},'trainlm');
                         %[neuronas de entrada y salida],
                         %{funciones de activación logarítmica
sigmoidal y lineal}
                         % Algortimo de aprendizaje
                         net.performFcn = 'mse';% error cuadrático
medio

                        %DIVISIÓN DE DATPS

                         net.divideFcn = 'dividerand';  % División
aleatoria
                         net.divideMode = 'sample';  % Divide up
every sample
                         net.divideParam.trainRatio = 80/100;
                         net.divideParam.valRatio = 10/100;
                         net.divideParam.testRatio = 10/100;

                         %PARÁMETROS DE ENTRENAMIENTO
                         net.trainParam.epochs = 1000; %Número
máximo de iteraciones
                         net.trainParam.goal = 0; %Error objetivo
                         net.trainParam.time = inf;%tiempo máximo de
entrenamiento
                         net.trainParam.min_grad = 1e-5;%Gradiente
mínimo
                         net.trainParam.lr = 0.06;%Mu tasa de
aprendizaje
                         net.trainParam.max_fail=30;% Iteraciones
despues de encontrar un mínimo

                         net = initlay(net);%inicialización de pesos


                         % ENTRENAMIENTO DE LA RED
                        %%%%%
                        [net,tr] = train(net,inputsV,target);

                        %GeneraciOn de pesos
                        %%%%
                        PesoEntradasF= net.IW;
                        PesosCapasF = net.LW;
                        PesosBiasF=net.b; %representa la constante


PesosCapas1=abs(cell2mat(PesoEntradasF(1,1)));

Pesos_Entrada(a,b)=struct('pesos',(cell2mat(PesoEntradasF(1,1))));

                        PesosCapas2=abs(cell2mat(PesosCapasF(2,1)));

Pesos_Salida(a,b)=struct('pesos',(cell2mat(PesosCapasF(2,1))));


PesosBias(a,b)=struct('pesos',(cell2mat(PesosBiasF(1,1))));
```

```
                        %Q(im) peso de la variable i a la salida m,
                        %Guardado en un arreglo matricial por
iteración

                         for I=1:1:k
                         num1=0;%numerador de la ecuaciOn en 0 para
acumular y generar la sumatoria
                         den2=0;%denominador de la ecuaciOn en 0
para acumular y generar la sumatoria
                             for i=1:1:k
                                  for j=1:1:l
                                       O=sum(PesosCapas1');  %O es
un vector sumatoria r,j

num1=(PesosCapas1(j,I)./O(j)).*PesosCapas2(j)+num1;

den2=(PesosCapas1(j,i)./O(j)).*PesosCapas2(j)+den2;
                                       Q(a,b,I)=num1./den2;
                                  end
                             end
                         end


                         format short %Banderas creadas para hacer
seguimiento de las iteraciones en caso de ser necesario

                         red(a,b,l) =
struct('cod',((10000)+(k*100)+(l)),'dir',tr);


                         %DEFINICIÓN DE VARIABLES PARA ANÁLISIS DE
CORRELACIÓN Y ERRORES
                         %%%%%%%%
                         output = net(inputsV);
                         trOut = output(tr.trainInd);
                         vOut = output(tr.valInd);
                         tsOut = output(tr.testInd);
                         trTarg = target(tr.trainInd);
                         vTarg = target(tr.valInd);
                         tsTarg = target(tr.testInd);

                          %Regresiones y R^2
                          mdl = LinearModel.fit(trTarg,trOut);
                          T=mdl.Rsquared.Adjusted;
                          R2T(a,b)=T;
                          mdl = LinearModel.fit(vTarg,vOut);
                          V=mdl.Rsquared.Adjusted;
                          R2V(a,b)=V;
                          mdl = LinearModel.fit(tsTarg,tsOut);
                          TS=mdl.Rsquared.Adjusted;
                          R2TS(a,b)=TS;

                         %analisis del error
                         mseT=mean((trOut-trTarg).^2);
                         mseV=mean((vTarg-vOut).^2);
                         mseTS=mean((tsTarg-tsOut).^2);

                         MSET(a,b)= mseT;
                         MSEV(a,b)= mseV;
                         MSETS(a,b)= mseTS;
```

```
                        Qdos(a,b) = 1-(mseV/mseT);

                        b=b+1;

                    end

          a=a+1;

        end
    end

end
```

ANEXO 2. PESOS VIENTO

Tabla 13. Pesos Viento

		Pesos por variables																								Bias Capa1	pesos salida	Bias salida
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25			
N. Neuronas	1	-0,82	-0,05	0,06	-0,05	0,15	-0,54	-1,33	-0,07	-0,24	0,16	-0,57	-0,36	-0,94	0,80	-0,69	1,13	-1,08	-0,57	0,73	-0,26	0,87	0,82	-4,23	6,58	-261,31	3,14	-0,84	-1,01
	2	0,46	0,86	-0,74	-0,68	0,17	0,16	0,23	-1,00	-0,71	-0,56	-0,24	0,93	-0,24	0,79	0,37	1,16	0,72	-0,65	-0,58	-0,36	0,80	0,82	-6,11	0,97	2171,56	-3,67	0,48	
	3	-0,77	-0,28	-0,34	0,14	-0,71	0,43	-0,63	0,11	-1,29	0,36	0,39	-0,64	-0,12	0,57	0,29	0,23	-1,27	-1,32	0,37	-0,65	0,08	0,55	-8,61	8,05	1690,72	2,39	-0,10	
	4	0,44	-0,55	-0,44	0,33	1,52	0,01	-0,53	0,68	0,95	-0,32	0,15	-0,75	0,56	-0,87	1,08	-0,17	0,52	-0,27	-1,13	0,14	0,32	-0,77	4,37	-5,94	424,34	-2,77	-0,75	
	5	-0,62	0,69	0,34	0,10	0,58	-0,68	-0,73	-0,84	0,33	-0,02	-0,60	0,71	-0,28	-0,16	0,34	-1,10	-0,28	0,72	0,55	-0,99	-0,18	-0,82	-8,61	7,95	-673,03	3,15	-0,27	
	6	1,05	0,83	0,10	-0,33	-0,07	0,08	-0,85	-0,22	0,06	-0,22	-1,00	-1,04	0,04	0,30	-0,79	1,24	0,65	-0,65	0,78	0,73	-0,37	0,28	8,09	-7,74	1330,97	-3,17	0,86	
	7	-1,01	0,69	-1,27	-0,40	-0,10	-1,11	-0,29	-0,66	-1,30	0,33	0,82	-0,06	-0,27	0,19	0,09	0,14	0,77	-0,02	-0,13	1,12	-0,37	-0,26	1,92	-8,20	41,52	3,24	-0,34	
	8	0,63	-0,22	-0,84	-0,08	0,33	-0,67	-1,11	0,18	0,77	-0,55	1,21	1,18	0,00	-0,56	-0,41	-0,52	0,51	-0,36	0,31	0,62	-0,29	-1,14	5,29	2,97	-1435,04	-2,45	-0,25	
	9	-0,83	0,19	-0,84	-0,07	0,66	-1,14	0,23	0,07	-0,36	1,36	0,40	-0,24	-0,10	-0,90	1,10	-0,55	0,39	-0,58	0,74	0,68	1,31	0,68	0,20	-0,43	1381,08	1,28	0,76	
	10	1,00	0,67	-0,24	0,17	1,22	0,40	0,07	0,84	-0,91	0,40	-0,33	-0,04	-0,48	0,28	0,68	-0,56	0,18	0,43	-0,89	-0,56	0,89	0,69	-9,47	8,36	1815,88	-2,95	0,84	
	11	-0,10	-0,62	-1,39	-0,41	0,40	0,00	-0,09	-0,48	-1,13	-0,75	-0,86	-0,14	-0,53	0,74	-0,66	-0,97	-0,68	-0,63	-1,11	0,41	0,08	0,21	10,20	5,79	1617,52	-4,10	0,38	
	12	-0,74	0,56	0,91	0,99	0,50	-0,42	0,31	-0,10	1,12	0,19	-0,45	-0,62	0,41	1,08	0,96	-1,02	-0,35	1,11	0,07	-0,68	1,19	0,49	0,64	-5,02	1004,06	1,47	0,33	
	13	0,50	-0,42	0,95	0,10	1,04	0,71	0,17	0,38	-0,40	-0,30	0,30	-0,73	0,33	-1,09	0,02	0,86	-0,50	0,59	1,00	0,23	-0,80	0,56	-5,65	3,57	1853,80	-2,21	1,68	
	14	-0,91	0,62	1,04	-0,05	-0,03	0,27	-0,09	-0,46	-0,69	-0,74	-0,62	-0,32	0,92	0,52	0,50	0,37	-0,29	-0,97	1,21	-0,77	0,24	-0,39	5,35	-7,37	438,59	1,44	1,23	
	15	0,86	0,00	0,95	0,48	-0,28	-0,70	1,36	0,10	0,50	-0,40	-0,41	0,13	-0,48	0,67	0,20	-0,01	-1,03	0,72	1,15	-0,39	0,51	-0,34	-10,30	-9,94	150,70	0,83	-0,79	
	16	1,07	1,33	-1,43	0,37	0,26	-1,08	0,71	-0,58	0,00	-0,60	0,55	0,82	-0,80	-0,12	0,95	0,83	-0,14	0,23	0,83	0,31	-0,87	-0,71	-5,76	7,01	-1384,97	-0,79	0,95	
	17	-0,39	0,71	-0,57	0,43	0,51	-0,45	0,59	0,13	0,94	0,64	0,04	-0,25	0,62	-1,10	-0,62	-0,64	-0,88	-1,03	-0,05	0,54	0,74	0,70	-8,42	-6,86	-52,83	2,60	-1,21	
	18	-0,34	0,08	-0,73	-0,30	0,43	-0,34	-0,27	-0,48	-0,49	-1,17	0,29	-0,90	0,18	0,73	-0,77	1,27	0,87	0,57	-0,91	-0,78	-0,32	0,86	8,92	5,27	-626,98	-0,39	-1,07	
	19	1,03	-0,07	1,19	-0,55	0,37	-0,15	-0,23	-0,30	0,72	-0,89	-0,20	-0,51	0,15	0,94	0,72	0,72	-1,02	0,42	0,28	0,75	0,42	-0,25	-10,77	-7,88	686,84	0,81	1,51	
	20	-0,68	-0,88	0,59	-1,05	0,30	-0,17	1,10	-0,76	0,63	-0,44	-0,75	0,43	0,33	0,57	-0,82	0,21	-0,61	0,97	-0,65	-0,84	0,90	-0,18	4,48	-4,88	308,72	0,46	1,50	
	21	0,71	-0,94	-0,46	0,09	0,92	-0,32	0,54	0,97	-0,93	0,56	0,96	0,50	-0,60	-0,85	0,53	-1,17	-0,45	0,52	0,17	-1,04	-0,38	0,59	-2,68	-6,06	489,41	0,28	0,31	
	22	-0,13	-0,02	-0,03	0,25	0,45	-0,09	-0,94	-0,18	-0,69	0,96	1,03	-0,89	0,30	-1,33	-0,10	0,87	-0,01	-0,97	-0,48	0,20	1,42	0,47	-1,62	8,80	-1303,68	0,14	1,69	
	23	-0,49	-0,71	0,71	-0,25	-0,55	-0,65	-0,50	-0,63	-0,92	0,37	0,85	0,10	0,53	-0,49	-0,46	-0,84	-0,86	-0,97	0,95	0,02	-0,53	0,98	-9,57	-1,96	490,98	1,01	-0,44	
	24	-0,80	0,45	-0,98	1,02	-0,44	-0,60	0,08	0,83	1,08	-0,51	-0,21	-0,78	0,11	-0,99	0,37	-1,32	-0,75	-0,54	0,73	-0,35	-0,01	0,25	4,31	1,37	13,43	-0,61	0,83	
	25	-1,04	-1,12	-1,18	0,08	0,49	-0,50	-0,34	0,62	0,53	0,97	-0,30	-0,16	-1,10	0,17	0,02	-1,10	0,08	0,25	0,58	1,13	-0,45	1,00	8,24	4,03	428,57	-1,66	-0,37	
	26	-0,62	0,64	0,18	0,13	0,25	-0,57	0,46	-0,95	-0,71	0,21	-0,36	-0,20	-0,45	0,07	-0,54	-0,65	-0,39	1,09	-0,58	1,00	0,96	-0,81	0,27	-3,26	1133,37	-0,54	1,12	
	27	-0,27	-0,28	-0,68	0,18	-0,08	0,01	0,61	1,10	0,43	0,56	0,67	0,49	-0,48	0,28	-1,02	-0,51	0,41	-0,36	-0,82	-1,15	-0,65	0,83	-8,46	9,79	708,34	-0,99	2,12	
	28	-0,91	0,41	-0,47	1,04	0,24	-0,83	0,33	-1,03	0,46	-0,51	0,61	0,54	0,30	-0,47	-0,69	0,83	-0,80	0,91	0,60	-0,94	0,06	0,72	7,88	4,51	729,14	-2,27	-0,59	
	29	-0,52	0,06	0,18	-0,12	-0,77	0,15	-0,46	0,93	1,11	-0,09	-0,77	-0,10	-0,78	-0,95	-0,73	0,75	0,32	1,16	-0,17	1,01	0,13	0,47	8,54	-2,59	-2087,54	-0,24	-0,79	
	30	0,60	-0,58	-0,36	0,49	0,02	-0,25	0,47	-0,31	0,65	0,31	0,50	0,91	-0,45	0,33	0,47	0,53	0,64	-0,62	0,04	-0,60	1,09	-1,21	4,04	-7,22	1845,61	0,36	-1,63	
	31	0,95	0,92	-0,83	1,00	1,21	0,91	-0,84	-0,04	-0,87	0,90	-0,46	-0,61	0,32	-0,41	0,35	0,31	-0,04	0,16	0,16	-0,22	0,95	-0,60	-9,10	0,26	1564,24	0,98	-0,96	
	32	-0,21	-1,18	0,44	-0,97	-0,29	-0,25	0,95	-0,20	1,23	0,09	0,12	-0,72	-0,79	0,83	1,23	-0,42	0,81	0,16	0,52	0,39	0,01	-0,22	-4,66	-6,32	-934,98	0,45	-0,14	
	33	1,30	0,74	-1,15	-0,48	-0,29	-0,88	0,07	1,11	0,29	0,11	-0,42	0,70	-0,55	0,33	0,44	0,34	-0,59	0,72	-0,71	0,75	-0,38	0,68	5,89	-1,14	-315,88	0,91	0,54	
	34	0,47	-0,32	-0,56	0,62	0,96	0,18	-0,56	0,63	-0,04	-0,92	-0,89	-0,21	0,96	-0,63	0,68	-0,92	0,60	0,48	0,57	0,14	-0,30	-0,87	3,84	4,05	2131,84	-0,48	1,55	

	35	0,43	-0,92	0,13	-1,26	0,73	0,06	-0,91	0,73	-0,77	-0,60	0,28	-0,85	-0,36	0,82	-0,31	0,24	-0,38	-1,01	0,20	0,76	0,03	-0,09	-3,93	10,45	463,59	0,54	-0,43
	36	0,99	-0,31	0,18	0,69	-0,27	0,98	0,61	-0,54	0,76	0,48	-0,06	0,22	-0,56	0,25	0,50	0,91	0,10	-0,67	-0,02	0,94	-0,68	-0,37	-0,12	-9,36	-829,64	3,11	1,07
	37	-0,73	-0,05	-0,17	0,07	0,90	-0,74	1,30	0,57	0,39	-0,29	0,25	0,07	0,95	-0,33	1,27	-1,48	0,36	-0,53	0,93	-0,70	-0,42	-0,23	5,69	6,69	1994,10	-4,31	-0,56
	38	-0,63	0,16	-1,16	0,07	-0,76	-1,10	0,58	-0,11	-0,35	0,11	-0,12	-0,90	1,02	-0,29	-0,23	-0,95	-0,48	-0,60	-0,44	-0,61	0,60	-0,31	0,46	4,33	2095,27	-3,66	-1,09
	39	1,14	-0,79	-0,11	-0,33	0,55	0,66	-0,06	0,57	0,69	0,87	0,92	-0,21	-0,87	-0,05	-1,00	-0,08	1,27	0,84	0,32	0,49	1,09	0,62	7,76	6,21	219,44	0,36	0,62
	40	-0,52	1,10	-0,01	-1,05	-1,02	-0,50	-0,80	-0,45	0,24	-0,18	0,31	-1,28	-0,07	0,21	-0,87	0,34	-0,25	-0,11	-1,03	-0,59	-0,95	0,97	2,69	-0,39	-229,96	-2,27	-1,18
	41	0,63	-0,45	1,33	-0,06	0,72	0,24	0,17	0,08	-0,47	1,73	1,11	-0,37	0,38	0,93	-0,77	0,82	-0,51	0,15	-0,11	-0,15	-0,32	0,15	-10,78	5,38	1891,91	1,78	-0,85
	42	0,81	0,40	0,27	0,17	-1,07	-0,87	-0,79	0,89	0,19	0,56	-0,23	0,36	0,11	1,07	-1,31	-0,38	-0,75	0,80	-0,07	0,15	0,31	0,15	-6,30	3,75	-1816,87	3,77	1,01
	43	-0,56	-0,34	-0,07	0,29	-0,28	-0,05	0,52	0,21	1,13	0,99	-0,51	-0,20	0,94	1,29	-0,81	0,28	-0,35	0,36	-0,98	-0,86	-0,12	0,80	-11,99	1,15	357,47	-1,89	-0,64
	44	-0,56	-1,18	-0,24	-0,35	0,22	0,89	0,91	0,92	0,59	0,16	-0,75	-0,54	-0,90	-0,62	1,03	-0,47	0,62	0,49	0,31	0,94	0,02	0,10	-6,35	0,30	-1552,12	-1,52	0,46
	45	-0,26	0,91	0,46	0,73	-0,22	-0,89	0,06	0,73	0,30	-0,84	0,43	0,04	0,66	-0,81	-0,63	-0,49	0,99	0,20	0,35	0,96	0,52	-0,46	5,96	-7,33	-1748,44	-1,83	0,42
	46	0,69	-1,04	0,76	-0,26	-0,49	-0,29	-1,11	-0,35	-0,75	0,55	-0,49	-0,65	-0,32	0,40	0,03	0,60	-0,01	1,06	0,81	1,08	0,62	0,05	-1,47	-0,04	-1763,59	4,22	-0,62
	47	-1,03	-0,65	-0,54	0,75	0,61	-0,14	0,03	-0,09	-0,84	-0,53	-0,08	0,66	1,15	0,73	0,00	-0,66	-0,59	0,21	0,47	-1,03	-0,42	0,90	1,10	-3,75	1813,88	-3,89	0,24

ANEXO 3. PESOS HELADA

Tabla 14. Pesos Helada

	Pesos por variables																													Bias Capa1	pesos salida	Bias salida
N. Neuronas	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31			
1	-0,95	-0,84	0,55	1,06	0,11	-0,60	0,99	0,74	-0,69	0,58	0,19	0,59	-0,47	-0,15	-0,22	0,73	0,25	-0,82	0,03	-0,55	-0,17	0,60	0,46	-0,13	0,23	0,20	-0,82	-0,95	-9,17	2,95	1013,62	2,68	0,24	-0,79
2	-0,06	-0,54	-0,60	-0,52	0,84	-0,47	0,86	-0,37	0,41	0,29	0,27	0,69	0,26	-0,31	0,52	0,36	-0,34	0,08	-0,85	-0,54	0,20	-0,35	-0,88	-0,77	0,01	-0,79	-1,00	-0,86	4,26	-8,50	497,55	-3,29	0,35	
3	-0,27	0,53	0,16	0,46	-0,24	0,34	0,84	-0,43	0,57	0,67	0,75	-0,65	0,24	-1,07	0,38	-0,42	0,60	-0,18	0,10	-0,27	-0,24	-0,38	-0,12	-0,74	-0,66	-0,28	0,83	0,98	-7,24	-16,97	1058,73	3,42	-0,89	
4	-0,74	0,43	0,48	0,61	-0,10	-0,35	0,59	-0,54	0,35	0,22	0,64	0,78	-0,85	0,17	-0,63	0,34	-1,14	-0,41	-0,31	-0,81	-0,92	0,55	0,35	0,97	-1,01	0,17	-0,10	0,11	3,94	-6,21	-841,88	3,63	0,00	
5	0,63	0,10	-0,81	-0,47	-0,15	-0,84	0,84	-0,77	0,84	-0,07	0,12	0,39	0,01	-0,25	0,91	-0,68	0,50	0,53	-0,68	-0,70	0,55	-0,21	0,84	-0,66	0,02	-0,04	0,19	0,56	10,92	10,65	509,36	-4,45	0,24	
6	-0,48	0,36	-0,08	-0,12	0,89	0,09	0,71	-0,79	-0,36	0,99	0,28	0,80	0,56	-0,43	-0,52	0,93	0,87	0,27	0,26	0,55	-0,87	-0,06	-0,17	0,12	0,32	-0,38	-1,00	-0,33	-0,07	-4,65	-739,29	3,42	-0,58	
7	0,66	-0,06	-0,50	-0,60	-0,96	-0,78	-0,33	-0,98	-0,61	-0,51	-0,30	0,14	-0,05	0,81	-0,06	0,91	0,02	0,98	-0,34	0,22	-0,30	0,39	-0,45	0,04	-0,75	0,11	-0,96	0,42	-2,25	13,35	-576,29	-2,56	-0,49	
8	-0,82	-0,92	-0,04	-0,76	0,40	0,98	0,54	0,14	-0,68	-0,63	0,70	0,46	0,78	0,32	-0,15	0,58	0,32	-0,34	0,74	0,83	0,03	-0,10	0,67	0,01	0,72	-0,01	-0,60	0,19	11,94	13,98	-506,82	1,28	0,00	
9	0,37	-0,44	0,19	0,13	-0,31	0,09	-0,58	0,46	-0,77	-0,22	-1,01	-0,54	-0,63	-0,22	-0,85	0,25	0,23	-0,31	0,72	-0,42	-0,51	0,11	-0,62	0,74	-0,43	0,76	0,87	0,68	-13,99	3,20	-1042,30	-0,52	-0,32	
10	0,73	0,34	0,00	0,43	0,22	-0,75	0,59	-0,40	-0,77	-0,63	0,35	0,94	0,96	-0,08	0,96	0,38	0,76	0,91	-0,08	0,19	0,96	-0,04	0,25	-0,28	0,40	-0,67	-0,29	0,87	5,30	4,32	-278,84	-2,40	-0,54	
11	-0,14	0,75	-0,42	-0,22	0,17	0,62	-0,84	0,85	0,23	-0,68	0,78	-0,85	0,10	-0,65	0,67	-0,03	-0,40	-0,32	-0,78	-0,82	-0,02	-0,74	-0,22	0,86	-0,61	-0,17	-0,77	0,27	3,97	3,86	-1095,97	2,35	0,71	
12	0,81	0,15	-0,52	-0,83	0,09	0,04	-0,83	-0,81	0,81	-0,40	0,39	-0,68	0,67	0,19	-0,38	0,56	-0,80	0,82	0,60	0,49	-0,73	0,55	0,73	-0,14	0,72	0,53	0,26	0,39	-5,09	12,77	-297,83	-1,90	0,40	
13	0,20	0,02	-0,52	-0,34	-0,34	-0,75	-0,57	-0,66	-0,65	-0,41	0,90	0,14	0,61	0,29	0,90	0,29	-0,13	1,10	0,09	0,51	1,18	-0,95	0,63	0,48	0,51	0,05	0,69	0,50	-7,01	13,34	13,13	-1,97	0,55	
14	-0,10	0,79	-0,11	0,09	0,93	0,24	-0,74	0,71	-0,74	-0,26	0,26	-0,83	0,81	0,02	0,38	0,93	-0,32	0,64	0,67	-0,57	-0,48	-0,64	0,17	0,89	0,63	-0,64	-0,82	0,41	-10,57	13,27	82,71	1,43	-0,05	
15	0,46	-0,11	-0,32	-0,26	-0,76	-0,48	0,09	0,78	0,46	-0,99	0,00	0,30	-0,10	1,00	-0,79	-0,86	-0,12	-0,64	-1,36	-0,74	-0,17	-0,53	0,75	-0,16	-0,13	0,61	0,84	0,60	3,68	-1,68	-259,71	-1,39	-0,68	
16	-0,80	-0,28	-1,14	-0,74	-0,54	0,78	-0,38	0,65	-0,98	-0,52	0,09	0,67	0,23	-0,12	0,48	0,14	-0,07	0,37	0,19	0,17	-0,19	0,86	0,17	0,16	-0,63	0,54	-0,07	-0,27	13,60	-14,45	-880,06	2,08	0,45	
17	-0,44	-0,47	-0,39	0,48	0,31	0,19	-0,97	0,74	-0,61	-0,12	1,09	0,80	0,43	0,35	-0,63	0,87	0,05	1,16	-0,26	-0,52	0,81	0,14	0,71	-1,11	-0,48	-0,56	0,35	-0,95	7,83	5,97	714,98	-0,27	-0,34	
18	0,91	0,23	-0,48	-0,44	-0,51	-0,74	0,24	1,08	-0,80	0,91	0,66	0,08	0,68	0,55	-0,10	-0,67	0,47	-0,09	0,68	0,09	0,26	-0,05	0,73	0,89	0,72	-0,08	0,16	0,54	-5,20	-14,61	807,01	-0,72	-0,68	
19	0,46	-0,40	-0,18	-0,39	0,20	0,33	0,57	-0,68	-0,48	0,76	0,07	0,62	-0,36	0,65	-0,15	-0,42	-0,86	-0,51	0,65	-0,54	0,47	0,37	-0,22	0,64	-0,65	-0,75	-0,09	0,28	-5,99	-13,61	-868,81	0,92	-0,39	
20	-0,80	0,26	0,21	0,17	-0,82	-0,56	-0,95	-0,59	-0,27	-0,22	-0,77	0,36	-0,77	-0,09	-0,59	0,21	-0,65	0,03	-0,73	-0,47	-0,78	-0,09	0,61	-0,97	-0,28	0,80	0,75	-0,36	-6,62	6,59	986,54	0,13	-0,48	
21	0,35	0,97	-0,53	1,26	-0,16	-0,24	0,63	0,04	0,85	-0,12	0,28	0,46	0,85	-0,17	0,15	-0,54	0,26	0,33	-0,50	-0,92	0,16	-0,02	-0,79	0,33	1,00	-0,35	-0,77	-0,81	-0,83	0,94	-348,45	-0,43	-0,06	
22	-0,05	0,37	0,88	0,20	-0,59	-0,06	0,27	0,32	0,77	0,65	0,64	0,27	-0,77	-0,69	-0,30	0,36	0,47	0,65	0,42	-0,87	0,05	0,02	-0,87	0,67	0,77	0,27	0,29	-0,72	-1,29	13,16	983,80	-1,95	1,00	
23	-0,65	-0,58	-0,65	0,10	0,09	-0,07	0,83	-0,08	-0,60	0,96	0,58	0,62	-0,94	-0,78	-0,79	-0,40	-0,60	-0,72	-0,02	0,51	0,61	0,40	0,45	0,81	-0,21	0,11	0,13	-0,74	9,53	-1,65	101,11	-0,25	-0,11	
24	-0,79	-0,63	-0,21	-0,49	-0,67	-0,72	-0,41	-0,54	0,02	0,69	0,78	0,66	-0,72	-0,18	-0,73	-0,10	-0,11	-0,20	0,64	0,31	-0,73	-0,36	-0,59	0,62	-1,03	0,97	-0,37	0,29	-0,15	10,04	669,77	-0,86	0,70	
25	-0,40	-0,63	-0,19	-0,90	-0,15	-0,62	0,98	0,19	-0,80	0,34	0,48	0,24	-0,87	0,40	-0,59	-0,11	0,88	0,59	0,70	-0,33	0,03	0,29	0,89	-0,15	-0,79	-0,55	0,68	0,69	1,38	-4,94	733,52	-0,24	0,29	
26	-0,93	0,74	0,08	0,07	0,78	0,21	0,99	0,80	-0,45	-1,08	-0,89	0,01	0,41	-0,09	0,29	-1,07	-0,80	0,30	-0,64	0,66	0,49	-0,04	0,43	0,37	0,17	0,39	-0,19	-0,36	-9,46	-3,47	398,88	0,63	-0,02	
27	0,18	0,73	-0,24	-0,67	-0,54	0,53	0,57	-0,84	-0,05	-0,19	-0,53	0,00	-0,65	-0,89	0,78	0,42	0,56	0,13	0,73	-0,41	-0,72	0,17	0,43	-1,11	-0,48	0,37	-0,52	0,62	4,73	17,34	-1033,71	-0,31	-0,10	
28	-0,80	0,34	0,61	-0,09	-0,76	0,01	-0,08	-0,54	-0,60	-0,26	0,03	0,00	0,73	0,88	0,48	0,21	-0,83	0,97	0,91	0,58	-0,45	0,12	-0,09	0,12	0,51	-0,91	0,91	0,42	1,73	-9,37	-866,54	0,92	0,42	
29	0,32	0,08	-0,84	-0,65	0,70	-0,79	-0,87	0,69	-0,24	-0,68	-0,38	-0,83	0,01	-0,81	0,52	0,73	-0,64	-0,35	0,51	0,35	-0,09	0,20	0,76	0,20	-0,51	-0,86	0,38	0,38	-6,38	-14,35	-124,96	1,60	-0,12	
30	-0,76	0,03	0,23	0,37	0,91	0,77	-0,84	-0,84	0,43	0,69	0,28	-0,31	0,59	0,19	0,84	-0,32	0,41	0,27	-0,74	-0,93	0,82	0,18	0,01	-0,50	0,47	-0,72	-0,46	-0,39	-7,85	16,69	622,71	-1,32	-0,66	
31	-0,72	1,18	0,70	0,12	-0,50	-0,97	-0,43	0,47	-0,78	-0,04	-0,70	-0,14	0,20	0,95	-0,44	-0,80	0,20	-0,29	0,22	0,84	-0,67	0,89	0,69	0,33	0,48	0,00	0,49	-0,16	7,53	13,60	-524,85	-1,44	0,25	
32	0,59	0,50	0,47	0,28	-0,72	-0,30	-0,61	-0,75	1,16	0,03	0,56	0,79	-1,00	0,29	-0,77	0,07	-0,19	1,30	-0,18	-0,38	0,82	0,15	0,07	-0,89	0,28	1,32	0,14	0,31	-14,69	10,67	-393,57	1,38	0,80	
33	-0,08	-0,51	-0,06	-0,78	-0,27	0,29	-0,93	0,07	0,61	0,68	0,58	0,82	-0,50	0,47	-0,58	-0,01	-0,05	-0,73	0,58	-0,82	-0,86	0,08	-0,24	-0,98	0,28	0,70	0,60	0,03	-10,70	5,74	977,47	-1,17	-0,15	
34	-0,72	-1,19	0,69	0,46	0,56	0,00	0,25	0,76	-0,43	0,55	-0,32	-0,68	-0,44	-0,10	1,00	0,71	-0,48	0,69	0,29	-0,06	-0,64	-0,35	-0,63	0,22	0,75	0,38	0,52	0,03	-13,39	-13,59	-41,75	0,68	-0,81	

	Pesos por variables																													Bias Capa1	pesos salida	Bias salida
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31			
35	0,37	1,25	-0,09	-0,04	0,08	-0,26	0,53	1,06	0,24	0,45	0,54	0,88	0,43	-0,59	1,00	0,44	-0,64	1,08	0,19	-0,27	-0,26	0,16	-0,29	-0,17	-1,01	0,36	0,63	-0,76	6,17	-2,54	-811,78	1,44	-0,68	
36	-0,52	0,22	1,00	0,10	-0,02	0,49	-0,46	-0,21	-0,08	0,01	0,00	-0,21	0,42	-0,21	0,39	1,03	-0,25	0,66	1,46	0,36	-0,42	1,23	0,34	0,62	-1,01	-0,12	-0,16	0,58	-7,88	-21,29	1022,26	-0,46	0,48	
37	-0,32	0,29	0,63	0,47	0,51	0,81	-0,68	0,14	-0,15	0,49	-0,82	-0,09	0,31	-0,06	1,32	0,40	-0,04	-0,44	0,32	0,43	0,12	-1,07	0,29	-0,51	-1,10	-0,75	-0,64	0,43	5,27	6,20	358,81	-2,33	0,76	
38	-0,13	1,10	0,88	-0,28	-0,86	-1,56	-0,03	-0,54	-0,98	0,41	-0,35	0,29	-0,71	0,46	-0,06	0,18	-0,16	0,63	-1,01	0,27	1,00	0,66	0,81	-0,52	-0,16	0,48	-0,38	-1,21	-9,73	5,67	-204,11	-0,90	-0,63	
39	-0,79	-1,13	0,78	0,81	0,38	-0,49	-0,01	0,48	-0,38	-0,32	-0,80	1,03	0,48	-0,43	-0,11	0,18	0,37	-0,33	0,14	0,60	0,06	-0,59	0,44	-0,76	0,70	0,13	1,13	-0,32	-6,68	-12,09	1035,02	-1,37	0,83	
40	-0,64	0,58	-0,09	-0,50	-0,03	0,12	-0,51	0,19	-0,40	0,30	-0,38	0,86	0,23	-0,42	-0,28	0,90	0,43	0,89	1,08	-0,38	-0,13	-0,71	-0,30	0,25	0,79	-0,55	-0,44	0,51	-13,09	14,66	916,89	-2,39	-0,52	
41	-0,78	0,18	0,50	0,43	0,49	0,18	0,83	-0,42	0,28	-0,50	-0,47	-0,44	0,38	0,82	0,43	-0,66	-0,65	0,47	0,86	-0,46	0,66	-0,40	0,09	-0,58	-0,17	-0,04	0,89	-0,81	10,85	-11,72	982,49	-2,77	-0,73	
42	0,58	0,88	-0,59	-0,33	0,58	-0,85	0,55	0,00	0,85	-0,74	0,11	-0,49	0,14	0,35	0,35	-0,33	-0,55	-0,94	0,32	-1,11	-1,15	0,12	0,32	-0,05	-0,71	-0,49	-0,54	0,51	-12,12	3,05	-405,71	2,90	1,03	
43	0,15	-0,71	0,09	0,23	-0,89	-0,30	-0,66	0,48	-0,28	-0,20	-0,75	0,91	-0,06	0,38	-0,31	-0,14	-0,63	0,51	0,70	0,10	0,74	0,09	-0,73	0,33	0,87	0,71	0,74	-0,52	-11,44	6,34	22,12	-1,69	-0,78	
44	0,24	-0,28	-0,24	-0,39	-0,07	-0,75	0,56	0,33	0,19	0,26	-0,95	-0,07	0,72	-0,63	-0,04	0,46	-0,75	0,52	0,11	-0,91	-0,75	0,18	0,22	-0,59	0,79	-0,83	-0,82	-0,68	0,26	14,52	-993,61	2,19	0,76	
45	-0,35	0,71	0,19	0,36	-0,45	0,62	0,17	-0,20	0,32	0,77	0,37	1,13	0,92	0,01	0,35	0,68	-0,48	-0,80	-0,78	1,33	0,30	-0,88	-0,09	-0,57	-0,07	-0,42	0,30	0,01	-10,56	6,71	294,96	-2,20	-0,87	
46	-0,51	0,71	0,91	0,25	-0,88	0,55	-0,17	-0,65	0,95	0,15	-0,31	0,67	-0,82	-0,51	-0,27	-0,45	-0,54	-0,37	0,55	1,04	0,09	-0,45	-0,52	-0,59	0,03	-0,61	-0,45	0,42	-8,78	2,21	-566,32	-1,52	-0,03	
47	0,05	0,39	0,23	-0,48	0,63	-0,61	-0,06	-0,37	0,52	-0,63	-0,34	-0,17	0,31	-0,40	-0,95	-0,66	0,89	-0,38	0,38	0,72	-1,54	0,16	0,89	-0,63	0,30	0,08	-0,87	0,33	3,37	-21,27	-526,98	4,00	-0,48	
48	0,62	0,17	-0,57	-0,53	0,16	-0,25	0,32	-0,58	-0,90	0,19	0,08	0,05	-0,25	0,84	-0,29	0,80	-0,05	0,33	-0,49	-0,91	0,51	0,75	-0,83	0,95	0,76	0,05	0,73	-0,21	-13,80	9,05	-882,69	3,87	1,08	
49	-0,17	-1,12	-0,05	-0,56	-0,10	0,97	-0,19	-0,48	-0,59	-0,05	-1,03	-0,15	0,21	0,24	0,72	1,07	0,30	0,65	-0,16	-0,38	-1,21	0,48	-0,30	0,83	0,01	-0,53	-0,56	-0,54	6,80	-4,87	-1026,99	-2,07	0,07	
50	0,52	0,65	-0,82	-0,23	-0,59	-0,42	-1,09	0,58	-0,93	-1,02	0,71	-0,36	0,12	-0,63	0,75	-0,90	-0,12	0,03	0,55	0,45	-0,40	-0,09	0,66	0,05	0,36	-0,73	-0,66	-0,05	-1,61	-1,89	-801,45	3,93	0,76	
51	0,21	0,73	-0,29	-0,50	-0,42	0,07	-0,21	0,07	-0,82	-0,60	0,40	0,78	-0,75	-0,12	-0,87	0,94	0,41	0,56	-0,32	-0,60	0,81	-0,71	-0,74	-0,48	-0,64	0,33	-0,01	0,69	-11,89	-10,40	-448,63	4,80	0,14	
52	0,41	-0,92	-0,46	-0,56	0,22	0,39	-0,11	-1,05	0,24	-0,63	-0,34	-0,44	0,53	-0,38	0,72	0,59	0,83	0,11	0,72	-0,09	-0,58	-0,69	-0,51	-0,52	0,48	-0,70	0,16	0,35	12,08	10,56	792,24	1,15	-0,92	