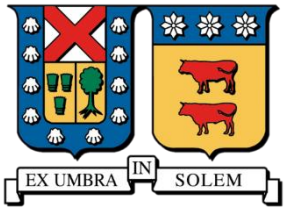
UNIVERSIDAD TÉCNICA FEDERICO
SANTA MARÍA
DEPARTAMENTO DE ELECTRÓNICA
VALPARAÍSO – CHILE



PRONÓSTICO DE DEMANDA DE AGUA POTABLE MEDIANTE REDES NEURONALES

NATALIA RODRÍGUEZ AEDO

MEMORIA DE TITULACIÓN PARA OPTAR AL TÍTULO DE INGENIERO CIVIL ELECTRÓNICO MENCIÓN CONTROL E INSTRUMENTACIÓN

PROFESOR GUÍA: ALEJANDRO SUÁREZ.
PROFESOR CORREFERENTE: DANIEL ERRAZ

JUNIO 2016

Agradecimientos

Primeramente, agradezco a mi familia por su paciencia, amor y apoyo incondicional durante toda mi etapa universitaria. Los amo.

Agradezco a mi pareja por ser siempre un apoyo y un consejero excepcional durante estos últimos 9 años.

Muchas gracias a mis amigos queridos que siempre estuvieron presente cuando necesité su ayuda.

Por último, agradezco a mi profesor guía por su buena voluntad, y estar siempre dispuesto a ayudarme.

Resumen

El presente estudio busca analizar la capacidad de redes neuronales artificiales para predecir la demanda de agua producida en las localidades de Los Andes, Calle Larga y Real Curimón de la provincia de los Andes.

La utilidad del proyecto recae en la obtención de la predicción de niveles de agua no contabilizada (pérdida de agua) en un mediano a largo plazo. La predicción de agua no contabilizada se obtiene de la diferencia entre los pronósticos de demanda de agua producida y demanda de agua facturada. La empresa sanitaria ya dispone del pronóstico de agua facturada, por lo tanto, el fin de este trabajo es entregar a la empresa el pronóstico de agua producida faltante.

Básicamente, las redes neuronales son una herramienta matemática que emula la estructura de una red neuronal biológica, con el fin de aprender patrones y asociar eventos a partir de un set de datos. La factibilidad de las redes neuronales como herramienta de pronóstico, está basada en estudios existentes en la literatura, donde son utilizadas para la predicción de datos en distintos contextos industriales. Particularmente, las predicciones de consumo de agua mediante redes neuronales ya han sido implementadas en el extranjero con excelentes resultados. Se utiliza *Matlab* con la librería *Neural Network Toolbox* para el desarrollo de la red neuronal predictiva.

La metodología de trabajo consiste en la evaluación de distintos aspectos del diseño de redes predictivas, tales como la inclusión de nuevas variables, linealidad de la red, diseño de distintas arquitecturas, desfase de entrenamiento, retardos de entrada, y frecuencia de datos. Cada configuración de red es simulada y sus predicciones son comparadas con datos reales. Según los resultados, el diseño de red óptimo corresponde a una red no lineal entrenada con datos mensuales de demanda de agua y temperatura máxima, la cual arroja un error promedio de predicción de *1.83%*.

Abstract

This study seeks to analyze the artificial neural networks ability to predict the water demand produced at localities Los Andes, Calle Larga and Real Curimón from Los Andes province.

The project usefulness lies on getting predicting levels of not recorded water (loss of water) in a mid to long term. Predicting unaccounted water is obtained from the difference between produced water demand and billed water demand forecasts. The sanitary company has already the billed water forecasting, therefore, the thesis goal is to give to the company the missing produced water prediction.

Basically, neural networks are a mathematical tool that emulates the biological neural network structure, in order to learn patterns and associated events from a data set. The feasibility of neural networks as a prediction tool is based on existing studies on literature, where they are used for data prediction in different industrial contexts. In particular, water consumption predictions using neural networks have already been implemented abroad with excellent results. *Matlab* with the *Neural Network Toolbox* library are used to develop the predictive neural network.

The working methodology consists in the evaluation of different aspects from predictive network design, such as the inclusion of new variables, network linearity, different architecture designs, training lag, entry delays and data frequency. Each network configuration is simulated with test data where their predictions are compared with actual data. According to results, the optimal network design corresponds to a nonlinear network trained with monthly data on water demand and maximum temperature, which generates an average prediction error of 1.83%.

Glosario

Las abreviaturas utilizadas en el presente documento tienen los siguientes significados:

- ANC *Agua no contabilizada.*
- MSE *Error cuadrático medio.*
- MAPE *Error porcentual absoluto medio.*
- MSPE *Error porcentual cuadrático medio.*
- INE *Instituto Nacional de Estadísticas.*

Índice

Índice de figuras

Índice de tablas

Introducción

En general, el ser humano siempre ha tenido la inquietud de pronosticar sucesos futuros que tienen relevancia en su vida. A lo largo de la historia, se han desarrollado una serie de técnicas y métodos para lograr este cometido. De ello, se obtienen una serie de herramientas de predicción con un sin número de aplicaciones y contextos diferentes, tales como, medicina, finanzas, transporte, clima, entre muchos otros.

Una de ellas, es la predicción de volumen de agua, la cual tiene una participación importante con respecto al abastecimiento de agua y el trabajo ingenieril que se debe llevar a cabo para ello. El diseño de una planta para abastecer una población determinada, se planifica en base a la predicción de consumo de agua de la población en cuestión. Asi mismo, una vez construida la planta, el pronóstico de producción y consumo de agua vuelve a ser relevante, pero esta vez, para la contabilización de pérdidas y planificación de producción.

Es en este último punto, es donde recae la utilidad del presente trabajo. En él se desarrollan métodos de predicción de demanda de producción de agua, en base al diseño de una red neuronal artificial. Una red neuronal artificial, es una herramienta de predicción, que, en este caso, es capaz de modelar la demanda de agua a través de sus datos históricos. La red es 'entrenada' con dichos datos, es decir, las variables de la red se adaptan según el comportamiento de los datos, para, posteriormente, ser capaz de generar una predicción mediante la simulación de la misma.

En el presente documento se presenta la metodología de diseño de la red neuronal, las decisiones tomadas y resultados obtenidos. Adicionalmente, se expone un contexto industrial y utilidad del trabajo para comprender la relevancia del proyecto para empresas sanitarias. Y, por último, para tener un mejor entendimiento de los conceptos nombrados a lo largo del documento, se presenta un marco teórico que describe las redes neuronales y describe los puntos más relevantes respecto a la aplicación que se le da en este trabajo.

CAPÍTULO 1

Contexto sanitario

En este capítulo se plantea el contexto sanitario del proyecto, en el que se dan a conocer una serie de puntos relevantes relacionados con el campo de la ingeniería sanitaria.

1.1 Sistema de agua potable

El agua potable, de forma generalizada, llega a los consumidores a través de un sistema sanitario que consiste en una obra ingenieril con la función de distribuir y potabilizar agua proveniente de fuentes naturales.

Esta red de abastecimiento está compuesta por una serie de etapas que procesan el agua cruda y la distribuyen a cada uno de los puntos de abastecimiento. A continuación, la Figura 1.1, nos muestra un sistema de agua potable.

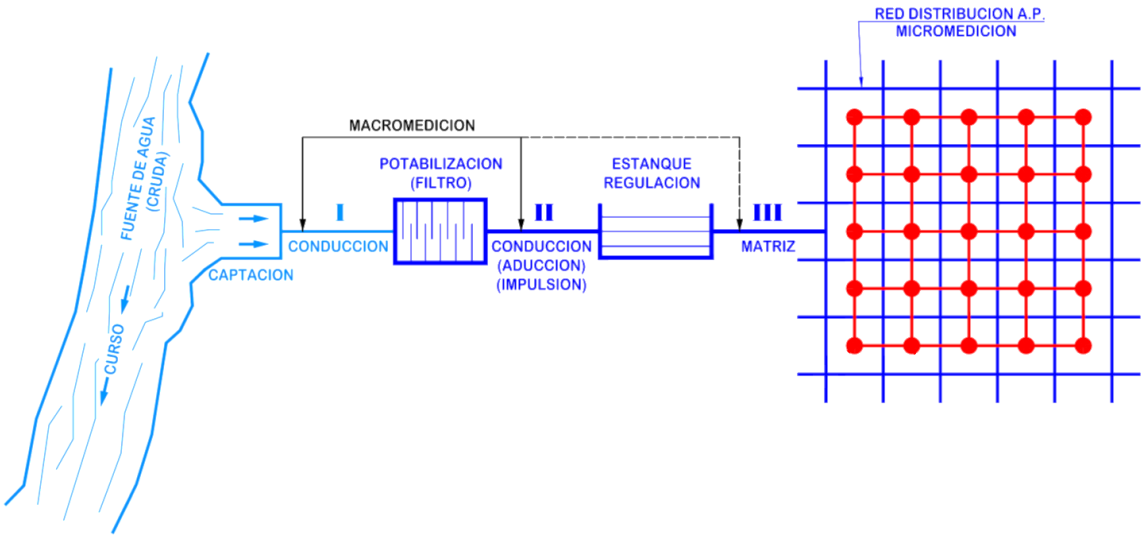


Figura 1.1 Sistema de agua potable

En general, las etapas de un sistema sanitario incluyen el tratamiento de agua potable y aguas servidas. La sección referente al sistema de aguas servidas, que incluye su recolección, tratamiento y descarga, es irrelevante para este trabajo, y, por lo tanto, omitida en este documento.

En cuanto al trabajo con agua potable, las etapas de un sistema de este tipo incluyen la captación de agua bruta, la producción, regulación y distribución de agua potable.

1.1.1. Captación

La etapa de captación corresponde a la extracción de agua bruta desde una fuente natural, la cual puede tener distintos orígenes. Algunos de los orígenes del agua bruta son; agua de lluvia almacenada, agua subterránea proveniente de manantiales, pozos o galerías filtrantes, agua superficial proveniente de ríos, arroyos o lagos, e incluso agua proveniente del mar (la cual requiere de procesos adicionales de desalinización).

1.1.2. Producción

La etapa de producción está subdividida por procesos específicos de potabilización, que corresponden a una serie de tratamientos de desinfección y filtración. Entre otras cosas, el tratamiento de agua incluye etapas de retención de material grueso y de material fino en suspensión, tratamientos químicos de decantación de materiales muy finos y desinfección en general.

1.1.3. Regulación

La etapa de regulación del sistema de agua potable está compuesta por una bomba de extracción que alimenta con agua tratada a un estanque de regulación (conocido popularmente como copa de agua), la cual tiene la función de compensar las variaciones horarias del consumo versus la frecuencia de funcionamiento de la bomba de extracción. Por lo tanto, la frecuencia de producción, que corresponde a la frecuencia de la bomba de extracción, es diferente de la frecuencia de consumo de agua.

Es importante notar que el volumen de producción diario es igual al volumen de salida del estanque de regulación en 1 día. Lo anterior, se debe a que la bomba de extracción entrega agua a caudal constante por ciertos periodos de tiempo al día, donde el volumen total de agua bombeada al estanque corresponde a los litros necesarios para el abastecimiento de 1 día. Por lo tanto, la demanda diaria de consumo de agua es igual a la demanda de producción diaria.

Adicionalmente, el estanque de regulación tiene la tarea de almacenar un volumen específico para casos de emergencia, de tal forma, de garantizar el abastecimiento de agua a los consumidores.

1.1.4. Distribución

La red de distribución corresponde al sistema de cañerías que llega a cada uno de los puntos de abastecimiento para los consumidores. Esta red se inicia en el estanque de regulación, y consta de estaciones de bombeo, tuberías, válvulas que sectorizan el suministro de agua en caso de rupturas y emergencias por escasez de agua, y, por último, de dispositivos de medición de volumen de agua en los puntos de abastecimiento.

1.2. Contabilización de agua potable

A continuación, se explican los elementos más relevantes para el entendimiento de la contabilización de agua que hacen las empresas sanitarias.

Como se menciona en la sección anterior, la red de distribución de agua, consta, entre otros elementos, de medidores de volumen de agua en cada punto de salida de la red de distribución. Este volumen de agua consumido por los clientes es llamado *facturación*, o *agua facturada*.

Por otro lado, como se explica en la sección 1.1.3, existe una demanda de producción de agua, que es distinta de la facturación. La diferencia entre el consumo de agua, o facturación, y el volumen de producción es llamada *agua no contabilizada (ANC)*.

El agua no contabilizada es un gran tema de consideración para las empresas sanitarias en el mundo, ya que finalmente las ANC son volúmenes de agua que se pierden y que generalmente corresponden a un porcentaje importante del agua producida por el sistema sanitario.

Las principales causas de perdida de agua son factores que contribuyen al deterioro de los sistemas de distribución de agua, y generalmente son del tipo, físicos, operacionales, y ambientales, entre otros.

Dentro de los factores físicos se encuentran el material, antigüedad, espesor y diámetro de tuberías, el tipo de unión entre ellas, su revestimiento y protección, y que existan tuberías de diferentes metales.

Entre los factores operacionales se encuentran las prácticas de operación y mantención mal realizadas, como reparaciones e instalaciones mal ejecutadas, presión interna de agua, o del transiente de agua que erosiona las tuberías, fugas, y calidad y velocidad del agua.

Algunos de los factores ambientales que pueden dañar el sistema de distribución de agua son el tipo de suelo, las aguas subterráneas, la ubicación de las tuberías y la actividad sísmica.

Por último, existen otros tipos de factores importantes, como lo es el robo de agua masivo y la construcción de inmuebles que puedan dañar las tuberías.

El trabajo de las pérdidas de agua, implica una inversión importante para la empresa sanitaria, ya que se trata de un enorme proceso de gestión, definición de estrategias y plan de inversiones que se traducen en un trabajo de identificación de causas de pérdidas, reparación de tuberías, implementación de estructuras, entre otros.

No trabajar en el control de agua no contabilizada, implica que las pérdidas existentes, debido a rotura de tuberías, aumenten, ya que estas se erosionan con el paso del tiempo. Lo mismo ocurre si la población comienza a crecer, ya que el agua que corre por las tuberías aumenta su velocidad para poder abastecer a más clientes, lo cual genera mayor roce en las paredes de los conductos, y, por lo tanto, un crecimiento en las grietas de la tubería.

1.3. Utilidad de la predicción

Para las empresas de servicios sanitarios en el mundo, el agua no contabilizada es un tema de contingencia, y en algunos países alcanza porcentajes vergonzosos. Se estima que un 34% de toda el agua mundial es agua no contabilizada [1]. Específicamente en Chile, los porcentajes de pérdidas superan el 40% para muchas empresas sanitarias. La Figura 1.2 nos muestra un gráfico que contiene los porcentajes de agua no contabilizada para las empresas que entregan servicios sanitarios en Chile [2].

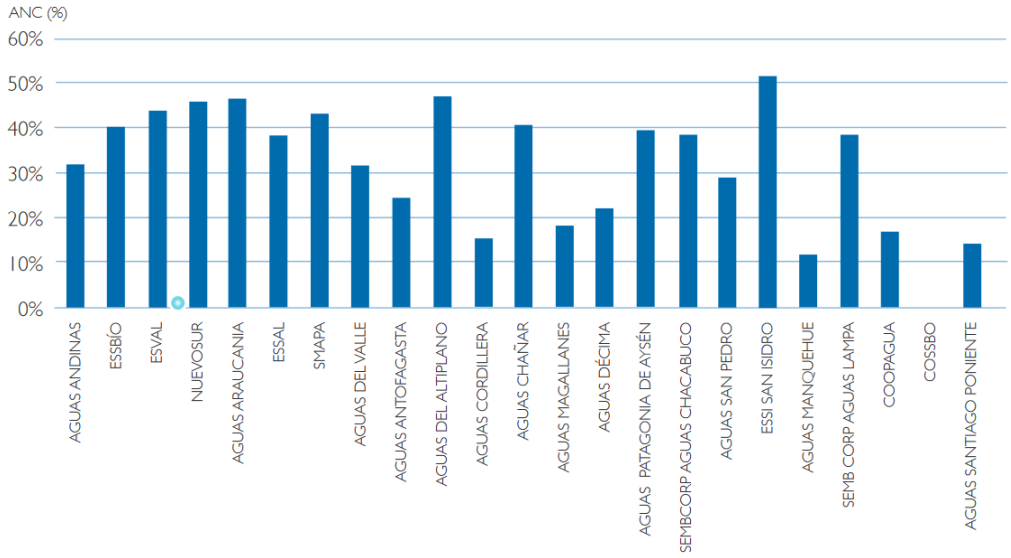


Figura 1.2 ANC en empresas sanitarias de Chile

Como se puede apreciar, los porcentajes de pérdidas van desde un 12% hasta un 52% aproximadamente, y en específico, la empresa sanitaria de interés, ESVAL, bordea el 45% de pérdidas.

En los últimos años, la empresa ESVAL ha estado bajo una nueva administración, la cual se ha encargado de trabajar en la eficiencia de la producción de agua potable, a diferencia de la administración anterior. Esto implica, entre otras cosas, tomar medidas para el control de agua no contabilizada.

La empresa, desde su nueva administración, ha comenzado la contabilización de agua producida para poder generar pronósticos de agua no contabilizada, y así, tomar decisiones respecto a la disminución de las pérdidas.

La proyección de agua no contabilizada se obtiene de la diferencia entre la proyección de agua producida y la proyección de agua facturada. La proyección de agua facturada, es totalmente factible de realizar para la empresa sanitaria, ya que existen métodos definidos y mucha información demográfica disponible, entregada por el gobierno a través del Instituto Nacional de Estadística (INE) y el CENSO. Es en este punto en que se refleja la utilidad de este trabajo, la empresa no dispone de la proyección de agua producida, por lo tanto, en pocas palabras, la finalidad de este

trabajo es entregar a la empresa sanitaria la proyección de agua producida, para que esta pueda obtener la estimación futura de agua no contabilizada.

Para trabajar en el pronóstico de agua producida, se deben tener en cuenta ciertos puntos importantes. Uno de ellos es generar predicciones a mediano o largo plazo. La medición de la facturación de agua, como se menciona en 1.2, es contabilizada por medidores que se encuentran en cada uno de los puntos de salida de la red de distribución. Esta medición es tomada por un funcionario de la empresa que va casa por casa registrando la medición de agua consumida por el cliente. A cada funcionario se le asigna un sector de trabajo, donde la medición total de la facturación le lleva un tiempo comprendido entre los 15 y 30 días. Adicionalmente, las mediciones mensuales de cada sector no se encuentran disponibles simultáneamente, lo cual implica, que no se puede tener una facturación total de un periodo específico, por lo tanto, para remediar el desfase en la medición, es conveniente generar predicciones de agua facturada en un mediano o largo plazo, y por consecuencia, la predicción de agua producida debe ser realizada en el mismo plazo.

Por otro lado, la predicción de agua producida en este trabajo está basado en los datos históricos de la misma, por lo tanto, la predicción no considera eventos no proyectables, como, por ejemplo, poblaciones flotantes (personas que residen temporalmente en el lugar, como veraneantes), construcción de muchos edificios en un corto periodo de tiempo, creación de una universidad, minera, o industria, etc. En general, la población debe tener un comportamiento estable.

El comportamiento de la población puede extraerse desde las mediciones de demanda de agua producida, ya que las pérdidas de agua solo amplifican la curva de mediciones y no cambian su forma. Por lo tanto, si la curva de demanda de agua producida es estable, la predicción de esta es válida.

Finalmente, se tiene en cuenta, que mientras mayor sea la población, más insensible es la curva de mediciones de demanda de agua producida ante eventos no proyectables como los mencionados anteriormente.

CAPÍTULO 2

Estado del arte

En el presente capítulo se exponen algunas alternativas de solución para el entrenamiento de las redes neuronales, y posteriormente una pequeña reseña de algunos trabajos relacionados con las redes neuronales como herramientas de predicción.

2.1. Planteamiento de alternativas de solución

La investigación de las posibles soluciones para el proyecto, se basa en la búsqueda de diferentes algoritmos de entrenamiento de los pesos sinápticos de la red a diseñar. Las alternativas presentadas son las 3 mejores opciones de algoritmos de entrenamiento disponibles en el software MATLAB, que corresponde a la herramienta utilizada para la predicción en este trabajo.

Cualquier concepto relacionado con redes neuronales mencionados a continuación, pueden ser consultados en el capítulo 3. Toda la información expuesta respecto a la descripción y análisis de cada alternativa de solución, puede consultarse en la referencia [3].

2.1.1. Algoritmo 'Scaled Conjugate Gradient'

En este algoritmo se realiza una búsqueda a través de direcciones conjugadas, que conllevan a una convergencia más rápida que en direcciones descendentes pronunciadas. En la mayoría de los algoritmos de gradiente conjugado, los cambios en los pesos, o *step size*, son ajustados en cada iteración. En este método se hace una búsqueda a través de la dirección del gradiente conjugado para poder determinar el cambio en los pesos que minimizan la función de desempeño a lo largo de la línea.

Para esta alternativa de solución, se propone específicamente, una cierta clasificación del algoritmo gradiente conjugado, llamado *Scaled Conjugate Gradient*. Este algoritmo fue diseñado para evitar el consumo de tiempo en la búsqueda del cambio de los pesos sinápticos, esto, porque es usual que este tipo de método utilice como algoritmo de búsqueda, la búsqueda de línea (line search), que es computacionalmente muy caro.

Este algoritmo es muy complejo de explicar en palabras simples, pero la idea es básicamente, combinar el enfoque del algoritmo *Levenberg-Marquardt* (algoritmo descrito más adelante), con el enfoque que presenta el gradiente descendente. Dentro de todos los algoritmos de entrenamiento de gradiente descendente, este puede ser el que más iteraciones necesite para converger, pero el número de cálculos que se realizan en cada iteración es significativamente menor, porque no se realiza una búsqueda de línea [3].

Ventajas y desventajas

- Rápida convergencia para procesos en general.
- En redes neuronales de capas numerosas, tiene muy buen desempeño y suele ser de rápida convergencia.
- Buen desempeño en problemas de aproximación.
- Tiene poco requerimiento de memoria.
- Para conseguir errores muy pequeños en la red, el algoritmo no tiene un buen desempeño en cuanto al tiempo en que demora en alcanzarlos.
- Como es un algoritmo complejo, es menos intuitivo buscar errores en la programación cuando se tiene inconsistencias en la red.

2.1.2. Algoritmo 'Quasi-Newton'

Los algoritmos de entrenamiento de Newton, son una alternativa al método de gradiente conjugado, para una optimización rápida. El cambio en los pesos, en el método de Newton es mostrado en la siguiente expresión:

$$\boldsymbol{x_{k+1} = x_k - A_k^{-1} g} \quad (\mathbf{2.1})$$

Donde $\boldsymbol{A_k}$ es la matriz Hessiana, correspondiente a las segundas derivadas de la función de desempeño en el valor actual de pesos y valores de sesgo (los valores de sesgo, son también llamados "off-set" o "bias").

Generalmente el método de Newton converge más rápidamente que el método de gradiente descendente. Desafortunadamente, es costoso para el computador, generar una matriz Hessiana para redes neuronales unidireccionales. Es aquí donde un nuevo método, basado en el algoritmo de Newton, destaca, ya que, en él, no se requiere el cálculo de segundas derivadas. Este método, corresponde a la segunda alternativa de solución, llamada, "*Quasi-Newton*". El método tiene la misma metodología que su antecesor, (Newton), con la diferencia de que este actualiza la matriz Hessiana por una aproximada para cada iteración del algoritmo, en la que solo utiliza información de primer orden de la función de desempeño. [3]

Ventajas y desventajas

- Generalmente converge en pocas iteraciones.
- El uso de este algoritmo, en procesos simples o redes acotadas, tiene un buen desempeño en cuanto a la convergencia del error.
- La computación requerida para este algoritmo se incrementa con el tamaño de la red.
- Este algoritmo requiere más computación y almacenaje que en el método de gradiente conjugado.

- Para una red muy grande, es mejor no preferir este algoritmo, ya que funciona mejor para redes pequeñas.

2.1.3. Algoritmo 'Levenberg-Marquardt'

Al igual que el método Quasi-Newton, este algoritmo fue diseñado para disminuir el tiempo de entrenamiento sin tener que calcular la matriz Hessiana. Cuando la función de desempeño tiene la forma de suma de cuadrados, como típicamente se ve en redes unidireccionales, la matriz Hessiana se puede aproximar de la siguiente manera:

$$\boldsymbol{H = J^T J} \quad (\mathbf{2.2})$$

Y el gradiente puede ser obtenido de la siguiente forma:

$$\boldsymbol{g = J^T e} \quad (\mathbf{2.3})$$

Con $\boldsymbol{J}$ igual a la matriz Jacobiano, que contiene las primeras derivadas de los errores de la red respecto de los pesos y valores de sesgo, y $\boldsymbol{e}$ correspondiente al vector de errores de la red. El cálculo del Jacobiano, es mucho menos complicado de calcular o computar, respecto de la matriz Hessiana.

El algoritmo *Levenberg-Marquardt* utiliza la aproximación, antes mencionada, de la matriz Hessiana para representar, en la siguiente expresión, la actualización de los pesos en cada iteración.

$$\boldsymbol{x_{k+1} = x_k - [J^T J + \mu\, I]^{-1} J^T e} \quad (\mathbf{2.4})$$

Cuando el escalar $\boldsymbol{\mu}$ es cero queda como el método de Newton, usando la aproximación de la matriz Hessiana. Cuando $\boldsymbol{\mu}$ es grande, se convierte en un método de gradiente descendente con un step size pequeño. El método de Newton tiene la ventaja de ser más rápido y más cercano a alcanzar el error mínimo, por lo tanto, el

objetivo, al elegir valores de $\boldsymbol{\mu}$, es acercarse a este método lo más rápido posible. El valor de $\boldsymbol{\mu}$ es disminuido después de cada reducción en la función de desempeño, y es incrementada solo en el caso en el que un paso tentativo incremente la función. Es de esta forma, que la función de desempeño se ve reducida en cada iteración del algoritmo. [3]

Ventajas y desventajas

- Este algoritmo es uno de los métodos más rápidos para entrenar redes de tamaño moderado del tipo unidireccional (sobre los cientos de pesos).
- Levenberg-Marquart tiene una eficiente implementación en el software de trabajo, Matlab.
- En procesos simples, su desempeño respecto a la convergencia del error, es alcanzado de forma rápida.
- A medida que aumenta el número de pesos y valores de sesgo, se requiere de mayores recursos computacionales y demora más tiempo en converger a un error deseado.

2.2. Selección de la alternativa de solución

A continuación, se lleva a cabo la elección de una de las alternativas presentadas anteriormente. Esta elección se basa en una serie de criterios enfocados mayormente en la eficiencia de los algoritmos presentados. El análisis consiste en estimar la relevancia de cada criterio aplicando una respectiva normalización para generar un sistema de "puntajes" en donde cada alternativa de solución podrá ser evaluada, y así finalmente concluir cual es la mejor opción.

2.2.1. Criterios de elección

Tiempo de convergencia

En el análisis de los distintos algoritmos de entrenamiento de la red, es importante definir una meta para el error. De este modo, se puede evaluar la rapidez con la que el algoritmo es capaz de llegar al nivel de error deseado. Es aquí donde es importante saber elegir el algoritmo con menor tiempo de convergencia, dependiendo de las características del proceso.

Resistencia a la cantidad de datos

En la etapa de desarrollo del proyecto, es necesario disponer de una serie de datos, correspondientes a las entradas y salidas del sistema, que posteriormente, serán utilizados para el entrenamiento de la red. Es por esto, que, a la hora de entrenar la red neuronal, el algoritmo debe tener un buen desempeño ante un set de datos numeroso, ya que esto se traduce en un alto número de pesos sinápticos y valores de sesgo que la red ajusta en su entrenamiento.

Capacidad de disminuir el error

Dependiendo de la complejidad del proceso a modelar y de la precisión que se necesite en los resultados de la predicción, puede ser necesario que se exija un bajo nivel de error a la red. Los distintos algoritmos de entrenamiento, tienen diferentes desempeños, dependiendo del error que se tenga como meta. Es por esto que se compararán las alternativas de aprendizaje de la red, según su desempeño para converger a errores pequeños.

Complejidad del algoritmo

A la hora de programar la red neuronal, se debe ser capaz de solucionar potenciales errores en el desarrollo del algoritmo. Es decir, que la complejidad del algoritmo influye cuando se quiere hacer una revisión, paso a paso, de la programación,

ya sea para encontrar algún error o inconsistencia, o cuando se desea hacer alguna configuración específica.

Costo computacional

En general, este criterio pretende comparar las alternativas de solución, en cuanto a los requerimientos de memoria y procesamiento, que se necesitan al ejecutar el entrenamiento de la red neuronal.

Para poder determinar la importancia de cada criterio, y la relevancia que tienen unos respectos a los otros, se genera una tabla con las ponderaciones relativas de cada criterio.

CRITERIOS	PORCENTAJE DE RELEVANCIA
Tiempo de Convergencia	30%
Resistencia a la cantidad de datos	25%
Capacidad de disminuir el error	20%
Complejidad del algoritmo	15%
Costo computacional	10%
Porcentaje total	**100%**

Tabla 2.1 Ponderación de criterios según relevancia

La ponderación general de los criterios, se basa, primeramente, en que todo algoritmo se evalúa según su rapidez de procesamiento, al hacer uso de ellos. Es por esto que el tiempo de convergencia, es el criterio con más relevancia.

Por otro lado, la resistencia a la cantidad de datos es importante, ya que, en el proceso de desarrollo del proyecto, se trabaja con un gran set de ejemplos, con los cuales se entrena a la red neuronal. Por lo tanto, es importante que la cantidad de datos utilizados no entorpezca el desempeño del algoritmo de entrenamiento.

Para el trabajo de predicción con redes neuronales, es probable que se necesite un alto nivel de exactitud en sus resultados, es por esto, que la capacidad de disminuir el error en el algoritmo es relevante.

Durante el desarrollo del proyecto, la complejidad del algoritmo de entrenamiento, como se mencionó antes, influye a la hora de entender y buscar algún error en la programación de la red, o bien, ante inconsistencia de los resultados.

Por último, es importante contar con los requerimientos computacionales que exija la programación del entrenamiento de la red. Se estima, que a pesar de ser un criterio relevante a la hora de comparar las alternativas y ejecutar los entrenamientos, los procesadores y espacio de memoria de los computadores de hoy en día, tienen recursos suficientes para todas las opciones. Aun así, es relevante en el desempeño del algoritmo durante su programación y ejecución.

2.2.2. Evaluación de alternativas

Para poder cuantificar el desempeño de las alternativas en cuanto a los criterios de selección expuestos, se crea un sistema de puntuación presentado en la Tabla 2.2.

SISTEMA DE EVALUACIÓN				
Muy deficiente	Deficiente	Suficiente	Bueno	Muy bueno
0 -20	21-40	41-60	61-80	81-100

Tabla 2.2 Sistema de puntuación para evaluación de criterios

A continuación, se procede a evaluar cada una de las alternativas, presentadas en 2.1, según los criterios de selección, y en base al sistema de puntuación presentado. La puntuación de cada alternativa está basada en los estudios experimentales expuestos en [3].

CRITERIOS	PUNTAJE	COMENTARIO
TIEMPO DE CONVERGENCIA	80	En general, para todo tipo de procesos, tiene muy buena convergencia al error meta.
RESISTENCIA A LA CANTIDAD DE DATOS	90	En este sentido, según experimentos mostrados en [3], este algoritmo es el de mejor desempeño respecto a los demás. En estos experimentos se ve una mejora en su desempeño, a medida que se aumenta la cantidad de datos ingresados a la red.
CAPACIDAD DE DISMINUIR EL ERROR	70	En redes neuronales pequeñas, su desempeño en cuanto a disminuir el error, es de término medio, (respecto de los otros algoritmos). Este desempeño presenta una pequeña mejora cuando la red es de mayor tamaño.
COMPLEJIDAD DEL ALGORITMO	30	Muy complejo, de muchas iteraciones.
COSTO COMPUTACIONAL	85	Este algoritmo tiene modestos requerimientos de memoria.

Tabla 2.3 Evaluación de algoritmo 'Scaled Conjugate Gradient'

CRITERIOS	PUNTAJE	COMENTARIO
TIEMPO DE CONVERGENCIA	60	A medida que el número de pesos y valores de sesgo aumentan, el tiempo de convergencia del error deseado aumenta considerablemente.
RESISTENCIA A LA CANTIDAD DE DATOS	35	El aumento en los datos entorpece el desempeño de este algoritmo.
CAPACIDAD DE DISMINUIR EL ERROR	80	El comportamiento de este algoritmo, en comparación a los demás, mejora a medida que el error deseado es reducido.
COMPLEJIDAD DEL ALGORITMO	85	Simples iteraciones, que incluyen aproximaciones de la matriz Hessiana.

COSTO COMPUTACIONAL	75	Sus requerimientos de memoria no son tantos como el algoritmo *Levenberg-Marquardt*, pero los requerimientos de procesamiento se incrementan a medida que el tamaño de red aumenta.

Tabla 2.4 Evaluación de algoritmo 'Quasi-Newton'

CRITERIOS	PUNTAJE	COMENTARIO
TIEMPO DE CONVERGENCIA	90	En general, posee un muy buen tiempo de convergencia en el entrenamiento de redes neuronales de todos los tamaños.
RESISTENCIA A LA CANTIDAD DE DATOS	60	A medida que la cantidad de datos aumenta, el algoritmo presenta una clara disminución en su desempeño. Esto se debe al incremento de pesos y valores de sesgo en la red.
CAPACIDAD DE DISMINUIR EL ERROR	95	Para redes de menor tamaño, su capacidad de disminuir el error se destaca mucho en comparación a los demás algoritmos. Mientras que el error meta va decreciendo en redes de mayor tamaño, el algoritmo sigue destacándose, pero en menor medida.
COMPLEJIDAD DEL ALGORITMO	85	Simples iteraciones, que incluyen aproximaciones de la matriz Hessiana, con matriz Jacobiano del error.
COSTO COMPUTACIONAL	70	El requerimiento de memoria de este algoritmo es mayor respecto al de las otras opciones de aprendizaje. Este requerimiento podría ser reducido a costa del tiempo de ejecución del entrenamiento.

Tabla 2.5 Evaluación de algoritmo 'Levenberg-Marquardt'

Finalmente se presenta la tabla con los puntajes resultantes de cada alternativa, según la ponderación de los criterios y evaluaciones presentadas anteriormente.

CRITERIOS	Alternativa 1	Alternativa 2	Alternativa 3
Tiempo de Convergencia	80	60	90
Resistencia a la cantidad de datos	90	35	60
Capacidad de disminuir el error	70	80	95
Complejidad del algoritmo	30	85	85
Costo computacional	85	75	70
Puntaje total	**73,50**	**63,00**	**80,75**

Tabla 2.6 Evaluación total de las alternativas

Mediante el sistema de puntuación y porcentajes de relevancia de los criterios propuestos, se llega a que la mejor opción para solucionar la problemática del entrenamiento de las redes neuronales es la tercera alternativa; algoritmo de entrenamiento ‘Levenberg-Marquardt’.

Durante el análisis se observa que la primera opción de algoritmo de entrenamiento, ‘Scaled Conjugate Gradient’, es otra alternativa muy factible de escoger como solución. El criterio decisivo para tomar la decisión de utilizar el algoritmo ‘Levenberg-Marquardt’, es el hecho de que el algoritmo de la primera alternativa posee una alta complejidad en la estructura de su algoritmo. Esto es relevante al programar y simular la red, ya que es ahí cuando el código presenta potenciales errores y/o inconsistencias que implican, en el caso de la primera opción de entrenamiento, una tediosa búsqueda del problema y una gran inversión de tiempo.

2.3. Trabajos relacionados

La tecnología de las redes neuronales son técnicas de cálculo que hoy en día son utilizadas en una extensa variedad de aplicaciones. Estas proveen resultados razonables en aplicaciones donde las entradas presentan ruido o las entradas están incompletas. Algunas de las áreas de aplicación son las siguientes:

Conversión texto a voz: Esto ofrece una ventaja sobre las tecnologías tradicionales, teniendo la propiedad de eliminar la necesidad de programar un complejo conjunto de reglas de pronunciación. Esto abre posibilidades de investigación y expectativas de desarrollo comercial.

Imagen: Compresión de imágenes, reconocimiento de caracteres y reconocimiento de patrones en imágenes.

Modelos económicos y financieros: Creación de pronósticos económicos, volúmenes de venta y niveles de producción. Las redes neuronales están ofreciendo mejores resultados en estos temas que los métodos convencionales.

ServoControl: Existen diferentes redes neuronales que han sido entrenadas para predecir el error que se produce en la posición final de un sistema de servomecanismo. Este error se combina con la posición deseada para proveer una posición adaptativa de corrección y mejorar la exactitud de la posición final.

Procesamiento de señales: Predicción a partir de datos, modelado de sistemas y filtrado de ruido.

Como ya se ha mencionado, el presente trabajo abarca el tema de la predicción de datos con redes neuronales. Por lo tanto, se procede a presentar algunos trabajos referidos a este tipo de aplicación.

Predicción de acciones

Consiste en el diseño de una red neuronal capaz de predecir la transición mensual del índice de precios de acciones a través de conocimiento experimental. En este trabajo se utilizan redes con arquitectura del tipo recurrente y entrenamiento con retropropagación del error entre el dato real y la predicción. Para la predicción de la transición de precios de acciones toman como entrada principal 8 años de datos económicos incluyendo la predicción del crecimiento mensual y la caída del índice del precio de acciones, esto, más algunos indicadores económicos relevantes. Los resultados muestran que las redes neuronales funcionan satisfactoriamente como herramienta eficiente en la predicción del precio de acciones [4].

Predicción de tráfico vehicular

En este trabajo se han utilizado redes neuronales recurrentes para una predicción a corto plazo del tráfico en una carretera. El objetivo es prevenir la congestión vehicular y controlar el acceso a la autopista. Para su desarrollo se usan datos con estimaciones de otros días en estado similar. Los mejores resultados se obtienen con redes multi-recurrentes, concluyendo que las redes neuronales entregan mejores resultados que los métodos estadísticos convencionales [5].

Predicción de tornado

El diseño de esta red neuronal está basado en datos obtenidos de un radar Doppler, que observa diferentes fenómenos que puedan llegar a originar un tornado. En palabras simples, la red desarrollada tiene el objetivo de diagnosticar cuales fenómenos detectados por el radar pueden llegar a producir algún tornado. Uno de los fenómenos más nombrados son las tormentas eléctricas, que no siempre anteceden a los tornados. El trabajo muestra técnicas de diseño, tales como, procedimientos para determinar el tamaño del conjunto de datos de entrenamiento y el número de nodos ocultos necesarios para el funcionamiento óptimo de la red. Finalmente, el autor

concluye que los resultados arrojados por la red neuronal escogida superan a los basados en reglas convencionales de predicción de tornados [6].

CAPÍTULO 3

Marco teórico

3.1. Introducción a las redes neuronales

Las redes neuronales son una herramienta matemática, capaz de aprender a partir de la "experiencia". Esta experiencia es aprendida por un gran número de nodos que finalmente, definen su función. Estos nodos, son elementos procesadores que se encuentran dispuestos en capas y operan de forma paralela en su aprendizaje.

Las redes neuronales artificiales pretenden emular la estructura de las redes neuronales biológicas con el objetivo de construir sistemas de procesamiento de información. Los elementos que la componen, se distribuyen de manera jerárquica y tienen la cualidad de adaptarse a los objetos del mundo real, tal como lo hace el sistema nervioso biológico.

A grandes rasgos, una red neuronal artificial ya diseñada, está compuesta de un número determinado de constantes, llamadas "pesos sinápticos", que adquieren un cierto valor luego de ser entrenadas con un conjunto de muestras específicas. Estas muestras se refieren a datos de entrada y/o salida, correspondientes a algún proceso con el que se quiera trabajar. Una vez entrenada la red neuronal, se puede considerar que esta es un sistema que modela el comportamiento de los datos utilizados.

3.2. Red neuronal biológica

Las redes neuronales son modelos artificiales que intentan imitar algunas de las características humanas, específicamente, las referidas a la habilidad de memorizar y asociar hechos. Es por esto, que podemos decir, que las redes neuronales son un modelo simplificado del cerebro humano.

El cerebro humano tiene la capacidad de procesar a una enorme rapidez gran cantidad de señales provenientes de los sentidos. Es capaz de mezclarlas o relacionarlas con información pasada y dar respuestas adecuadas, aún en situaciones nuevas. Lo más destacable es la habilidad de aprender a entregar información o respuestas sin la necesidad de instrucciones explícitas.

Todo esto, ha inquietado a los académicos por muchos años. La búsqueda de la creación de herramientas computacionales capaces de realizar tareas con cierta inteligencia y autonomía, ha sido una constante en sus investigaciones.

Para tener una buena noción del sistema nervioso biológico, existen 12 billones de células nerviosas en el cerebro humano que se encuentran conectadas unas con otras para transmitir pulsos de información. Cada una de estas células nerviosas o neuronas, tienen entre 5000 y 60000 conexiones dendríticas para comunicarse con otras neuronas. Las neuronas biológicas están compuestas, entre otras cosas, por una entrada y una salida llamadas dendritas y axón, respectivamente. La conexión dendrita-axón es unidireccional y llamada sinapsis. A continuación, la Figura 3.1 nos muestra una neurona, en la que se representa lo antes descrito.

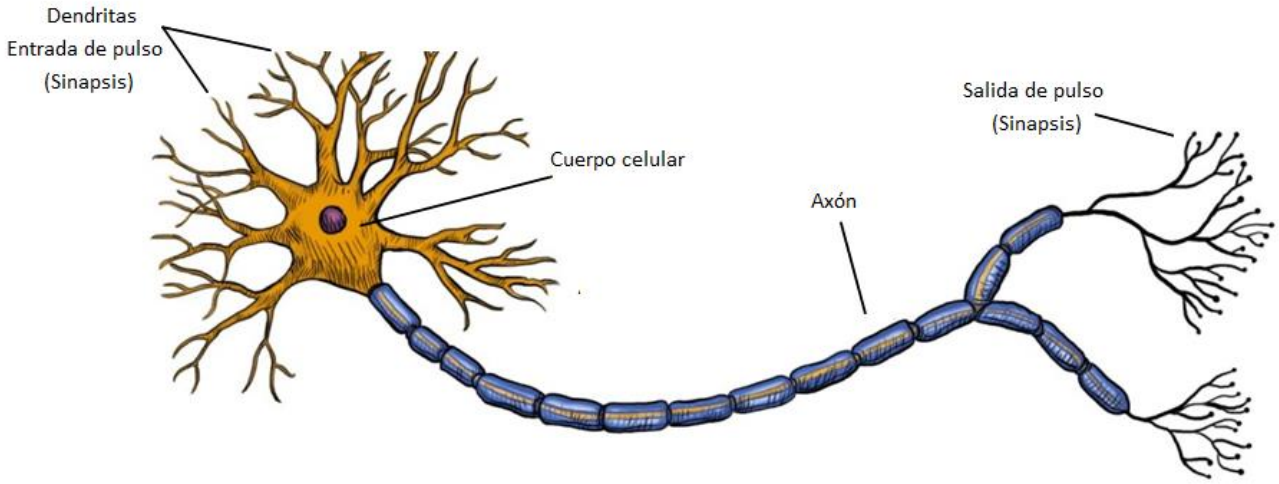


Figura 3.1 Célula nerviosa

Para un mejor entendimiento de la comunicación entre neuronas, es importante mencionar que las neuronas son eléctricamente activas, donde los flujos de corriente son originados a partir de una diferencia de potencial entre las membranas celulares de las neuronas. En pocas palabras, un impulso nervioso al interior de la neurona corresponde a un cambio de voltaje en la membrana celular de ésta. Cuando este impulso nervioso llega a los terminales del axón se produce una liberación de una sustancia química, llamada neurotransmisor, la cual se mueve por el espacio sináptico para llegar finalmente a la dendrita de otra neurona. Cuando los neurotransmisores llegan a la dendrita de la siguiente neurona pueden excitar o inhibir la neurona receptora, es decir, sinapsis con peso positivo al activar o peso negativo al inhibir.

Es importante aclarar que la señal de información que viaja por la neurona cambia su naturaleza de eléctrica a química o viceversa según la etapa de comunicación en la que se encuentre. El impulso que viaja por el axón de la neurona es de naturaleza eléctrica, mientras que en el espacio sináptico entre el terminal del axón de la neurona emisora y la dendrita de la neurona receptora es de naturaleza química.

En el proceso de la recepción de información a la neurona, las señales son sumadas o restadas, según si la neurona se excita o inhibe, al cambio del potencial de la membrana de la neurona. Si este total llega a los 10[mV], se dispara con una cierta frecuencia o tasa de disparo. Incluso, este disparo puede contener uno o más impulsos que se propaguen por el axón.

En términos biológicos, el aprendizaje de la red neuronal se genera en el momento en que el impulso es o no aceptado por la neurona receptora, es decir, el aprendizaje se genera en la sinapsis. La codificación del aprendizaje, se ve cuando los neurotransmisores inhiben o excitan la neurona, lo cual cambia el grado de influencia que ciertas neuronas tienen sobre otras. Por lo tanto, la estructura de la red y las conexiones neuronales son las que finalmente caracterizan el conocimiento de la red.

3.3. Red neuronal artificial

Básicamente, las redes neuronales artificiales corresponden a una herramienta matemática que pretende imitar la estructura "hardware" del cerebro emulando el sistema nervioso humano. Todo esto, permite que los algoritmos numéricos del sistema neuronal le den a la red la capacidad de aprender patrones y asociar eventos, de tal manera, que se logre representar un comportamiento de inteligencia. Al fin y al cabo, las neuronas artificiales representan modelos matemáticos que simulan el comportamiento de neuronas biológicas con tal de crear inteligencia artificial.

3.3.1. Neurona artificial

La neurona artificial es un elemento de proceso básico de la red, la cual genera una única salida, mediante el ingreso de un vector de datos proveniente del exterior o bien, de la salida de otras neuronas. Estas unidades de proceso, están compuestas por un núcleo, en donde los datos ingresados a la neurona son procesados matemáticamente para su posterior salida. Los vectores de entrada, son ponderados por un número determinado de constantes adaptables llamadas pesos sinápticos, los cuales enlazan las neuronas, reflejando la intensidad y relevancia de cada neurona. Una vez que las entradas han sido ponderadas, son sumadas en el núcleo de la neurona, para luego ser puestas a prueba por una función de activación, que determina la magnitud de la salida y la excitación o inhibición de la neurona.

A continuación, la Figura 3.2 nos muestra un modelo de una neurona artificial, que representa lo descrito anteriormente.

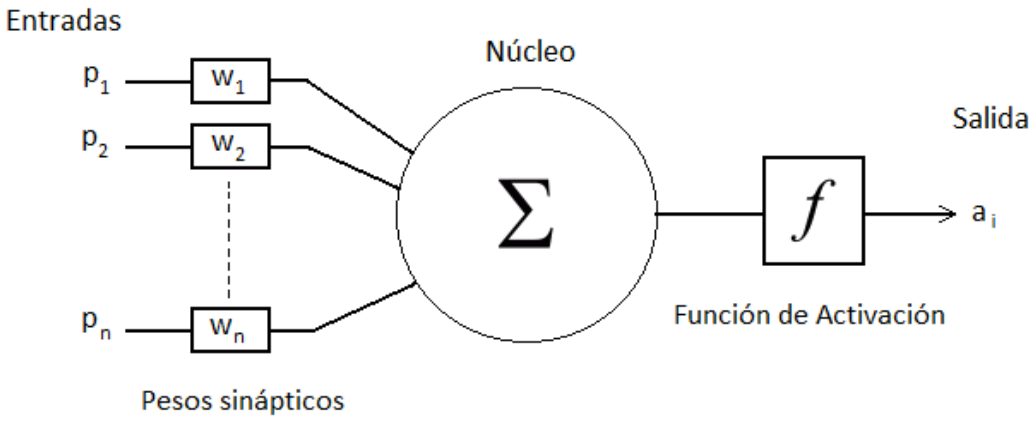


Figura 3.2 Modelo de neurona artificial

La salida de la neurona, puede significar una de las entradas de otra neurona o bien puede tratarse de la salida del sistema total. La naturaleza de la salida de cada neurona, o nodo, depende de la arquitectura del sistema y de la ubicación de la neurona en la red.

3.3.2. Arquitectura

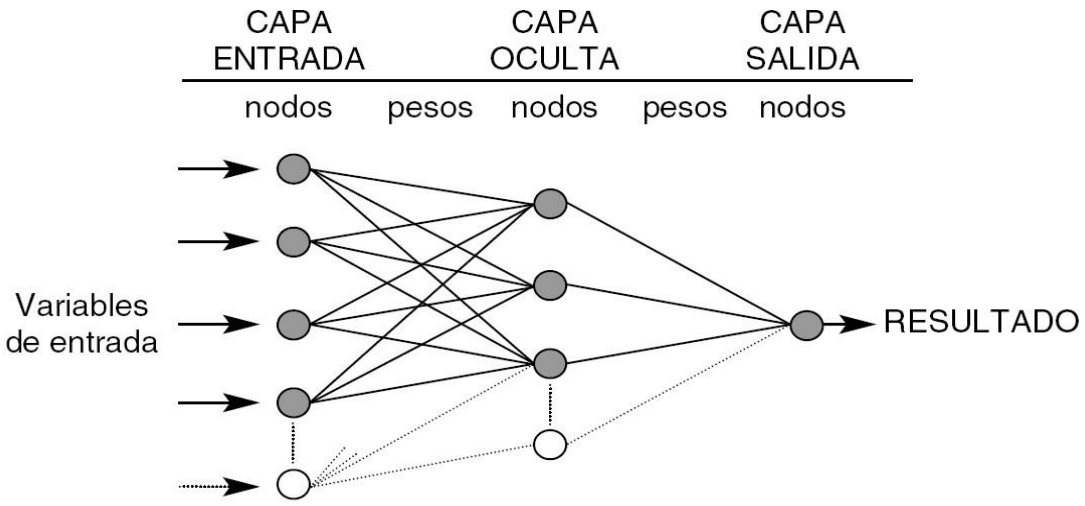


Figura 3.3 Arquitectura de una red neuronal

La arquitectura de un sistema neuronal corresponde a la distribución y disposición de las neuronas en una red. En general las neuronas suelen agruparse en unidades estructurales llamadas capas, de las cuales, como se muestra en la Figura 3.3**,** existen tres tipos.

El primer tipo, corresponde a la *capa de entrada*, la cual se encarga de recibir los datos o variables de entrada procedentes del exterior. El segundo tipo son las *capas ocultas*, de las cuales puede existir una o más capas, y en ellas se realiza el procesamiento interno de la red. Finalmente, la tercera capa, la *capa de salida*, proporciona la respuesta de la red a los estímulos de entrada de acuerdo a sus nodos y pesos sinápticos.

Otro punto importante, respecto a la arquitectura de una red neuronal, es si la red presenta, o no, ciclos internos en el seguimiento del "camino" que los datos recorren en la red. Cuando se trata de una red en la que no se puede trazar una línea de una neurona a sí misma, la red corresponde a una red unidireccional o *feedforward*. De lo contrario, se trata de una red del tipo recurrente o *feedback*.

Las redes neuronales pueden tener elementos retardadores en cada una de sus capas o neuronas según el diseño de la red. Cuando alguna de las neuronas en una red tiene la habilidad de recordar valores pasados, es de decir, poseen retardos, se habla de una *red neuronal dinámica*. En cambio, cuando ninguna de las neuronas del sistema recuerda valores pasados, es decir, no posee retardos, se habla de una *red neuronal estática*.

3.3.3. Funciones de activación

Las funciones de activación en una red neuronal corresponden a aquel "umbral" por el que se pone a prueba el valor de la sumatoria de las entradas de la neurona multiplicadas por sus pesos correspondientes. La señal de salida de la neurona, o bien

de la red (dependiendo si la neurona está o no en la capa de salida), será aquel valor que indique la función de activación.

En las redes neuronales, las funciones de activación pueden ser distintas por cada capa de neuronas. En el caso de redes neuronales con conexiones hacia adelante, o *feedforward*, que funcionan con algoritmos de entrenamiento de *retropropagación*, (definidos más adelante) deben tener funciones de activación diferenciables, continuas y crecientes.

Las funciones de activación más utilizadas en redes de este tipo, son las mostradas en las siguientes figuras.

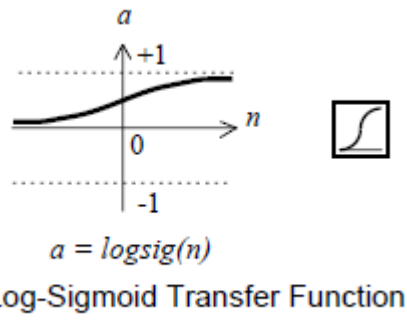


Figura 3.4 Función de activación logarítmica-sigmoidal

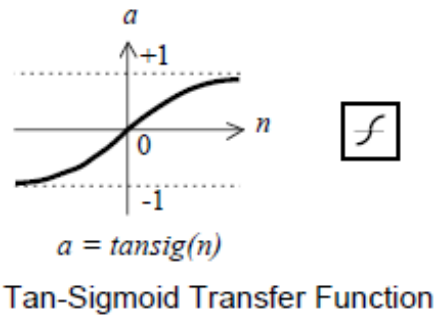


Figura 3.5 Función de activación tangencial-sigmoidal

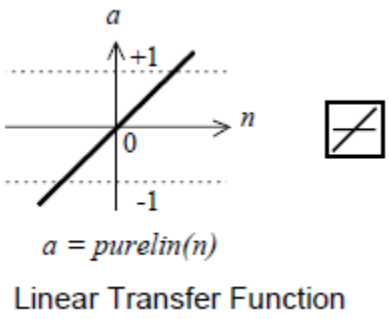


Figura 3.6 Función de activación lineal

De las alternativas presentadas, la función logarítmica-sigmoidal, toma el valor de entrada, que puede tener cualquier valor entre mas y menos infinito, y genera una salida en el rango entre 0 y 1. Esta función se puede utilizar para procesos continuos en los que se tengan valores positivos como salida.

En el caso de la función tangencial-sigmoidal, la salida se encuentra en el rango de valores entre -1 y 1, dependiendo del signo de la entrada. Esta función generalmente es utilizada en los casos en que se esté trabajando en el punto de operación de algún proceso o sistema.

Por último, en el caso de la función lineal, su uso es útil en los casos en los que las neuronas de las últimas capas de la red sean sigmoidales. Las salidas entregadas, por las últimas capas se encuentran en un rango limitado, por lo tanto, al colocar la función de activación lineal en la última capa de la red, la salida podrá tomar cualquier valor sin ser limitado por un rango.

3.3.4. Entrenamiento

El concepto de entrenamiento de una red neuronal, se puede definir como el proceso en el que el sistema neuronal “aprende” a lograr el objetivo de la aplicación de la red, es decir, que, a partir de un conjunto de entradas, la red neuronal entrega un

conjunto de salidas deseadas y/o consistentes. Dicho aprendizaje se logra ajustando los pesos de la red, mediante el proceso de entrenamiento.

Es importante mencionar, que existen varias modalidades para entrenar una red, respecto al conjunto de datos que se le provean. El *entrenamiento supervisado* es el tipo de entrenamiento en el que la red es dotada de datos de entrada con sus correspondientes salidas. El conjunto de datos de entrada es ingresado a la red, para luego comparar las salidas obtenidas con las salidas reales. Esto genera un error, entre ambas salidas, que modifica y actualiza el valor de los pesos hasta que dicho error converja a un nivel aceptable.

También existe el entrenamiento del tipo *no supervisado*, en el que el sistema solo es provisto de datos de entrada, y entrena los pesos sinápticos basado solamente en los valores de entrada y salida de la red en progresión.

La tercera y última modalidad de entrenamiento según el conjunto de datos que se le entregue a la red, es el *entrenamiento por refuerzo*, el cual se encuentra justo en medio de las dos modalidades anteriores. En este tipo de aprendizaje se le presenta a la red un conjunto de datos de entrada, y solo se le indica si la salida obtenida es o no aceptable o correcta, sin proporcionarle la salida real.

3.3.4.1. Backpropagation

El algoritmo de retropropagación o *backpropagation*, corresponde al método de entrenamiento más utilizado en redes neuronales del tipo unidireccional o feedforward. Este algoritmo es un método de aprendizaje supervisado que, junto a métodos de optimización, como el método de gradiente descendente o conjugado, propaga el error de la red desde la capa de salida, por las neuronas de las capas siguientes, hasta la capa de entrada, adaptando sus pesos sinápticos para finalmente minimizar el error.

Los algoritmos básicos de backpropagation, van ajustando los valores de los pesos sinápticos de la red, en donde la función de desempeño, es decir, la función que cuantifica el error que va cometiendo la red, va decreciendo más rápidamente.

El método de gradiente descendente es uno de los métodos de optimización más utilizado para algoritmos básicos de entrenamiento de redes neuronales. Este se lleva a cabo mediante la definición de la función de error antes mencionada, correspondiente a la diferencia entre la salida deseada y la salida que comete la red neuronal, en función de sus pesos sinápticos. Su objetivo es encontrar la configuración óptima de pesos, de tal forma que se llegue al mínimo global de la función de error, o bien, a un mínimo local suficientemente bueno, según las metas de desempeño que se establezcan en el diseño de la red. En el Anexo A, se describen las ecuaciones que explican este método.

En general, el método de gradiente descendente, utilizado por sí solo, como algoritmo de entrenamiento de tipo backpropagation, suele ser lento para problemas prácticos o de datos muy numerosos.

Por otro lado, el método de gradiente conjugado, corresponde a uno de los métodos de optimización más utilizados en algoritmos de alto rendimiento de backpropagation. Al igual que el gradiente descendente, este método tiene el objetivo de encontrar un mínimo global o local de la función de desempeño para la obtención de los pesos sinápticos óptimos de la red. En el Anexo B se puede consultar la descripción de este método.

Los algoritmos de entrenamiento presentados en la sección 2.1 son algoritmos de alta eficiencia que utilizan como base métodos de optimización numérica como los mencionados anteriormente. En ellos se presentan algunas variaciones y/o aproximaciones para mejorar su eficiencia y rapidez de convergencia.

CAPÍTULO 4

Desarrollo del proyecto

De forma general, este capítulo presenta todos los aspectos relacionado a la ejecución y desarrollo mismo del proyecto.

En él se describe mayormente la metodología de trabajo, explicando paso a paso todas las directrices tomadas en el desarrollo del trabajo, incluyendo el tratamiento de datos, diseño de la red, estimación de futuros resultados e implementación.

4.1. Metodología de trabajo

4.1.1. Diseño de la red neuronal

A continuación, se desglosa el detalle del diseño de la red neuronal predictiva en todos sus aspectos.

4.1.1.1. Selección de la variable

Teniendo en cuenta que el objetivo general del proyecto es realizar un pronóstico de una demanda de agua de una cierta localidad, debemos definir cuáles son los recursos necesarios para lograr este cometido.

De forma general, para realizar una proyección, es necesario adquirir un conjunto de observaciones sobre los valores que tome una determinada variable, que en nuestro caso corresponde a demanda de agua. En la literatura, este conjunto de observaciones es llamado *"Serie de tiempo"*.

Los datos que componen una serie de tiempo se pueden comportar de diferentes formas a medida que el tiempo pasa. Pueden, o no, presentar una tendencia, ciclos, una forma definida o aleatoria, estacionalidad, etc.

Las series de tiempo tienen un enfoque netamente predictivo, y, por lo tanto, los pronósticos son basados en el comportamiento histórico de la variable [7].

Para generar predicciones en una red neuronal, se precisa de datos que ayuden a la red a aprender el comportamiento de los datos a predecir. No es necesario que los datos para entrenar la red sean exclusivamente datos históricos de demanda, sino más bien, cualquier tipo de información que afecte o tenga que ver de alguna manera con la forma o tendencia de los datos. En este caso, se considera que la temperatura ambiental de la localidad afecta el comportamiento de consumo de agua en los clientes, es por esto, que esta variable es considerada en la predicción.

Por lo tanto, para este trabajo, se precisa de una serie de tiempo de datos de demanda de agua y temperatura en una misma frecuencia de muestreo.

4.1.1.2. Recolección de datos

Teniendo en consideración lo anterior, se debe procurar disponer de los datos requeridos.

Los datos utilizados en este trabajo fueron entregados por la empresa de servicios sanitarios ESVAL de Valparaíso. La información entregada corresponde a mediciones de demanda diaria de agua potable en litros/día, recibida por una copa de almacenamiento para posterior distribución a los clientes. Esta copa de almacenamiento abastece las localidades de Los Andes, Calle Larga y Real Curimón de la provincia de los Andes.

La serie temporal de demanda diaria de agua potable corresponde a las mediciones del periodo comprendido entre enero de 2005 hasta septiembre de 2015, los cuales fueron entregados por la empresa. Estos datos son entregados en 3 planillas Excel, correspondientes a las mediciones de la salida de los sistemas de producción de agua potable Bellavista, El Sauce y Miraflores, que, en conjunto, hacen el

abastecimiento de la copa de almacenamiento de salida. Según la empresa, los datos de cada sistema de producción sumados equivalen a la salida de la copa durante 1 día.

Por otro lado, se dispone de datos de temperaturas diarias máximas y mínimas de las mismas localidades del periodo comprendido entre enero de 2005 hasta marzo de 2015. Esta información es recopilada por la empresa y entregada para el desarrollo de este proyecto.

4.1.1.3. Pre-procesamiento de datos

Las redes neuronales son sistemas capaces de identificar patrones en la información, por lo tanto, la representación de los datos es muy importante en el diseño de una red con buenos resultados.

Como se mencionó en el punto anterior, los datos son entregados en planillas Excel, que contienen, dentro de otros antecedentes, la información necesaria para el pronóstico de demanda de agua. La empresa indica que algunos de los archivos scripts, encargados de registrar las mediciones de agua en el archivo Excel, falla de vez en cuando, y, por lo tanto, es común tener errores en la información de las planillas.

Ante esto, se requiere una corrección de la información para poder presentar un buen set de datos de entrenamiento a la red. Los datos son trasladados a un nuevo archivo Excel, de forma ordenada, según fecha de antigüedad y sistema productivo correspondiente (Bellavista, El Sauce y Miraflores).

Primeramente, se realiza una revisión de los datos, para corregir valores errados, o bien, inexistentes. Para ambos casos, se rellena la celda correspondiente con el valor medio entre las demandas del día anterior y el día siguiente. Cuando es necesario corregir más de una celda consecutiva, se procede a llenar con el valor medio entre las demandas de la misma fecha del año anterior y el año siguiente.

Luego de ordenados, completados y corregidos los datos en el nuevo archivo Excel, se deben normalizar, para que el rango de trabajo de la red sea más pequeño. Esto, debido a que cuando la red neuronal trabaja en una zona más acotada, demora menos en su entrenamiento y simulación.

La normalización de datos obedece a la ecuación 4.1, en la cual, y_n corresponde a los datos normalizados, y_{real} corresponde a los datos de la planilla Excel, e y_{media} corresponde al punto de equilibrio de los datos, que, en este trabajo, es definido por el promedio de todos los datos.

$$y_n = \frac{y_{real}}{y_{media}} \qquad (4.1)$$

En la sección de implementación de la red en el software correspondiente, se explica cómo se extraen los datos del archivo Excel y se ejecuta lo anteriormente explicado.

4.1.1.4. Definición del conjunto de entrenamiento

El conjunto de entrenamiento de la red neuronal corresponde al grupo de datos, con los que se entrenan los pesos sinápticos de la red para aprender el comportamiento de estos mismos. Por otro lado, se debe tener presente el conjunto de prueba, que corresponde al conjunto de datos con los que se evalúa la precisión del pronóstico de la red neuronal. Es muy importante, que los datos del conjunto de prueba no participen, por ningún motivo, en el entrenamiento de la red, ya que, de lo contrario, no se podrían evaluar los resultados del pronóstico.

Como se menciona en la sección anterior, se dispone de 3 conjuntos de datos o entradas, de los cuales 3925 son datos de demanda de agua diaria, 3742 datos de temperatura máxima diaria y 3742 datos de temperatura mínima diaria. En el caso de utilizar los 3 conjuntos de datos para el pronóstico, o bien, demanda de agua y temperatura máxima, según se defina más adelante, se debe usar una misma cantidad

de datos por entrada, por lo tanto, en ambos casos se utilizan 3742 datos por cada variable. En el caso de utilizar solo las mediciones de demanda de agua para la predicción, se utiliza la totalidad de datos disponible de esta variable.

La proporción entre los conjuntos de datos de entrenamiento y prueba, será de un 80% y 20% respectivamente. Por lo tanto, los datos del conjunto de entrenamiento representan 2993 mediciones y los datos del conjunto de prueba, los restantes 749 datos (en el caso de utilizar todas las variables). Se considera que esta proporción es adecuada en ambos casos, es decir, un 80% de los datos, son suficientes para representar al conjunto de datos totales para el entrenamiento de la red, y, por otro lado, el 20% de los datos, los cuales representan un poco más de dos años de información, son suficientes para la evaluación de la red.

4.1.1.5. Selección de la arquitectura

En esta sección se explican las distintas consideraciones a tomar en el diseño de la red neuronal, según los datos disponibles, la presentación de los datos a la red, y la linealidad (o no linealidad) de las funciones de activación en cada capa.

4.1.1.5.1. Red dinámica y estática

Primeramente, hay que considerar que la red a diseñar debe ser del tipo dinámica como se definió en 3.3.2, ya que la red debe ser capaz de recordar valores pasados. La entrega de los datos a la red, puede ser configurada con un cierto número de retrasos o *delays* en la primera capa de la red.

Para este trabajo se entregan los datos a la red de dos maneras, para luego escoger la de mejores resultados. La primera forma, es, como se mencionó antes, con retrasos en la entrada de la red, de manera que, al presentarle el conjunto de entradas a la red en el entrenamiento, esta sea capaz de definir cada set de ejemplos con un número

(x) de entradas, correspondiente al tamaño de la primera capa, y una salida, que se encuentra en la posición $(x + 1)$ de los datos, es decir, al siguiente valor en el set de datos.

La segunda forma de proporcionarle los datos a la red, es generando manualmente cada set de ejemplos para el entrenamiento de la red, sin el uso de retardos en la capa de entrada, generando la misma organización que el set de ejemplos de la opción anterior. En esta alternativa, a pesar de que el uso de redes dinámicas son indiscutiblemente el tipo de redes que debe utilizarse para este tipo de trabajo, requiere del uso de una red estática que se comporte como una red dinámica, es decir, a la red estática se le ingresan los datos desfasados manualmente, de tal forma, que al igual que la otra alternativa, la red recuerde datos pasados, solo que esta vez, no lo hace por sí misma, si no gracias a la forma en que los datos son ingresados para su entrenamiento.

En la etapa de implementación se realizan ambas estrategias, y según sus resultados, se escoge la mejor opción.

4.1.1.5.2. Linealidad

Otro aspecto a decidir en el proceso de diseño de la red, es si utilizar una red del tipo lineal o no lineal. Una red no lineal corresponde a una red que, en sus capas interiores, posee funciones de activación del tipo sigmoidales (no lineales), como las ilustradas en la sección 3.3.3. Este tipo de redes son utilizadas para el caso en el que se trabaje con datos no lineales. Por otro lado, una red lineal, es la red que posee solamente funciones de activación lineal, también ilustrada en el capítulo mencionado. Y al contrario de una red no lineal, son idóneas para el trabajo con datos lineales.

Como el conjunto de datos corresponde a un comportamiento de consumo de agua de personas, y no a un proceso definido o conocido, es difícil asegurar la naturaleza de los datos, (lineal o no lineal). Es por esta razón que, en la implementación del proyecto se diseñan redes de ambas naturalezas para definir cuál es la mejor opción.

De todas maneras, al tratarse del comportamiento de consumo de personas, se espera que los datos respondan mejor a una red no lineal.

4.1.1.5.3. Uso de temperaturas

En el diseño de la arquitectura de la red, se analiza otra variable importante, esta es si ocupar los datos de temperaturas máximas y mínimas diarias. Se considera que la temperatura podría ser un factor en el comportamiento del cliente respecto al consumo de agua, por lo tanto, en la etapa de implementación se diseñan 3 opciones, considerando un pronóstico solo con datos históricos de demanda de agua, otra considerando la demanda y temperaturas máximas diarias y por último una opción considerando toda la información; demandas, temperaturas máximas y temperaturas mínimas. En el próximo capítulo se evalúan sus resultados y se escoge la mejor opción de trabajo.

4.1.1.5.4. Frecuencia de datos

Adicionalmente, es importante considerar la frecuencia de los datos a utilizar en el entrenamiento y simulación de la red. Esto se refiere a si los datos son presentados a la red de forma diaria, quincenal, mensual, anual, etc. Tomando en consideración la cantidad de datos disponibles para el trabajo de predicción, se decide trabajar con datos diarios y mensuales.

4.1.1.5.5. Desfase en entrenamiento de la red

Finalmente, como último aspecto importante en el diseño de la arquitectura de la red neuronal, se entrena a las redes con un desfase variable entre los datos de entrada y de salida en el momento del entrenamiento de la red. Esto se realiza para que la red aprenda desfasadamente el comportamiento de los datos, es decir, si se le entrega una cantidad (x) de datos a la entrada, correspondiente al tamaño de la primera capa, a la

salida se le entrega el dato $(x + d)$, así la red adquiere la habilidad de pronosticar un dato que se encuentre dentro de (d) días o meses en el futuro.

Como el resto de los puntos mencionados en este capítulo, se muestran los resultados de los experimentos del desfase de datos y uso de información diaria y mensual en la sección de resultados del próximo capítulo.

4.1.1.6. Criterios de evaluación

En esta sección se explica cómo se evalúan los resultados predictivos entregados por el sistema de redes neuronales diseñadas para el pronóstico de demanda de agua.

De forma general, cada una de los diseños de redes neuronales nombrados en la sección anterior son evaluados con los mismos criterios. Para medir la eficiencia de cada red, se utiliza el cálculo del error cuadrático medio (MSE) entre el conjunto de datos reales y las predicciones obtenidas por la red. En la literatura y en muchos trabajos sobre predicción con redes neuronales, es común el uso de este parámetro, por lo cual se considera un estimador apropiado para la evaluación de los resultados.

La ecuación 4.2, muestra la expresión que representa el cálculo del error cuadrático medio, en donde Y_i corresponde al vector de n valores reales, y P_i representa el vector de n predicciones.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(P_i - Y_i)^2 \qquad (4.2)$$

Adicionalmente, para una mejor claridad en la medición del error de la predicción de la red, se utiliza como segundo criterio de evaluación el error porcentual cuadrático medio (MSPE), que no es más que una expresión del error cuadrático, dividido por el valor real, como un porcentaje. Al tratarse de un porcentaje, es más fácil de entender el error en la predicción.

La ecuación 4.3, muestra la expresión que representa el cálculo del error porcentual cuadrático medio, en donde Y_i corresponde al vector de n valores reales, y P_i representa el vector de n predicciones.

$$MSPE = \frac{100}{n} \sum_{i=1}^{n} \frac{(P_i - Y_i)^2}{Y_i} \qquad (4.3)$$

Finalmente, como último criterio de evaluación, se escoge el parámetro de estadística llamado error porcentual absoluto medio (MAPE). Se considera el uso de este criterio, porque es el factor más intuitivo a la hora de examinar resultados y analizar gráficos comparativos entre los datos reales y pronosticados. Al no presentar cálculos cuadráticos, el error arrojado, tiende a ser de mayor valor en comparación a los criterios antes explicados, por lo cual, esta estadística genera una relación más cercana, entre el valor del criterio y la visualización del pronóstico en gráficos. Por lo tanto, se considera que este criterio amplía la evaluación de la red y el análisis de los resultados.

La ecuación 4.4, muestra la expresión que representa el cálculo del error porcentual absoluto medio, en donde Y_i corresponde al vector de n valores reales, y P_i representa el vector de n predicciones.

$$MAPE = \frac{100}{n} \sum_{i=1}^{n} \left| \frac{(P_i - Y_i)}{Y_i} \right| \qquad (4.4)$$

4.1.1.7. Implementación

La implementación de este trabajo se lleva a cabo por completo en un mismo software, *Matlab*, con el cual se realiza la recopilación de datos, diseño, entrenamiento y simulación de la red neuronal.

Primeramente, para claridad del lector, se explica de manera simplificada cómo trabajar con este software, y así, tener un mejor entendimiento de lo expuesto a continuación. En caso de dudas, se puede consultar el Anexo C, donde se encuentran los códigos diseñados para la programación de redes neuronales predictivas.

Al programar en Matlab, todas las actividades se generan en una pantalla de comandos llamada *command window*, en donde datos y funciones pueden ser llamadas para su ejecución.

Es importante, que todos los archivos que se estén utilizando se encuentren en una misma carpeta. Matlab da la opción de visualizar mediante una pequeña ventana, el directorio del archivo en el que se trabaja y su contenido. Esta ventana es llamada *current folder*. Por ejemplo, en nuestro caso, el archivo Excel del que se extraen los datos está en la carpeta de trabajo, y es visualizada en la ventana current folder.

Los datos, y variables de todo tipo, con los que se trabajen, es decir, que hayan sido creados en una misma experiencia con el software, están almacenados en un espacio de trabajo llamado *workspace*.

Por último, para crear, trabajar y modificar una función determinada creada por el usuario, existe una ventana llamada *editor*. En ella se crean funciones que deben ser almacenadas en la carpeta de trabajo mostrada en current folder, para así, poder llamarla y ejecutarla desde la ventana de comandos, command window.

4.1.1.7.1. Datos

Para el uso de la información en el entrenamiento y simulación de la red, necesario que los datos, luego de ser corregidos desde su archivo Excel, sean normalizados como se explica en la sección en 4.1.1.3.

La implementación de la entrega de datos al workspace de Matlab, se lleva a cabo mediante una función que extrae los datos desde el archivo Excel y los ordena en

forma vertical, para generar un vector columna con el total de datos. Esta función es capaz de generar datos diarios, o bien, mensuales, según se especifique en el argumento de la función al llamarla desde command window.

Los datos son extraídos desde el archivo Excel mediante el comando *xlsread*, el cual tiene como argumentos, el nombre del archivo, el nombre de la hoja de trabajo, y el rango de celdas en las que se encuentra la información requerida, tal como se muestra en la figura siguiente.

A continuación, la Figura 4.1 nos muestra una sección del código de la función de adquisición de datos, en la que se observa la extracción de los datos de demanda de agua desde el año 2005 al 2015 del sistema de producción El Sauce. La función tiene como argumentos la especificación de la frecuencia de datos, es decir, si se desean

```
% EXTRACCIÓN DE DATOS ESVAL DESDE ARCHIVO EXCEL

function [LosAndesDDA, LosAndesTmax, LosAndesTmin, Promedio] = Datos_Esval(argFrec, temp)

if (temp == '1')  % 1
        % DEMANDA
        % El Sauce
        ElSauce2005 = xlsread('Recopilacion datos ESVAL.xlsx', 'El Sauce', 'D7:D371');
        ElSauce2006 = xlsread('Recopilacion datos ESVAL.xlsx', 'El Sauce', 'J7:J371');
        ElSauce2007 = xlsread('Recopilacion datos ESVAL.xlsx', 'El Sauce', 'P7:P371');
        ElSauce2008 = xlsread('Recopilacion datos ESVAL.xlsx', 'El Sauce', 'V7:V372');
        ElSauce2009 = xlsread('Recopilacion datos ESVAL.xlsx', 'El Sauce', 'AB7:AB371');
        ElSauce2010 = xlsread('Recopilacion datos ESVAL.xlsx', 'El Sauce', 'AH7:AH371');
        ElSauce2011 = xlsread('Recopilacion datos ESVAL.xlsx', 'El Sauce', 'AN7:AN371');
        ElSauce2012 = xlsread('Recopilacion datos ESVAL.xlsx', 'El Sauce', 'AT7:AT372');
        ElSauce2013 = xlsread('Recopilacion datos ESVAL.xlsx', 'El Sauce', 'AZ7:AZ371');
        ElSauce2014 = xlsread('Recopilacion datos ESVAL.xlsx', 'El Sauce', 'BF7:BF371');
        ElSauce2015 = xlsread('Recopilacion datos ESVAL.xlsx', 'El Sauce', 'BL7:BL96');
```

Figura 4.1 Comando 'xlsread' para extracción de datos

datos diarios o mensuales, y una variable que indica si se desean las matrices de datos de temperaturas.

La misma operación es realizada para los datos de demanda de agua del sistema de producción Bellavista y Miraflores, y para los datos de temperaturas máximas y mínimas, si es que estas variables son utilizadas.

Como se mencionó antes, todos los vectores anuales de datos, ya sean diarios o mensuales, son concatenados en un mismo vector de forma vertical, para generar solo un vector columna. Esto se lleva a cabo mediante el comando *vertcat*, como se muestra en la Figura 4.2.

```
ElSauce = vertcat(ElSauce2005, ElSauce2006, ElSauce2007, ElSauce2008, ElSauce2009, ...
Bellavista = vertcat(Bellavista2005, Bellavista2006, Bellavista2007, Bellavista2008, ...
Miraflores = vertcat(Miraflores2005, Miraflores2006, Miraflores2007, Miraflores2008, ...

Tmax = vertcat(Tmax2005, Tmax2006, Tmax2007, Tmax2008, Tmax2009, Tmax2010, Tmax2011, ...
Tmin = vertcat(Tmin2005, Tmin2006, Tmin2007, Tmin2008, Tmin2009, Tmin2010, Tmin2011, ...
```

Figura 4.2 Comando 'vertcat' para concatenar datos

Luego de generar estos vectores columna de datos, es preciso, en el caso de datos diarios, sumar los 3 vectores de demanda de agua de los distintos sistemas de producción. En el caso de pedir datos mensuales, se deben agrupar los datos, de manera que cada 30 datos se genere 1 dato del vector de valores mensuales. Posteriormente, al igual que en el caso de los datos diarios, se deben sumar los datos de los 3 sistemas de producción.

Obtenidos los vectores finales de datos reales, se procede a normalizarlos, dividiendo cada elemento de cada vector por el valor medio de cada uno. A continuación, en la Figura 4.3 se muestra un extracto del código que realiza la suma de los vectores de datos de cada sistema de producción, alternando cada 30 posiciones, de modo que se generen los vectores de datos mensuales. Esto seguido de la normalización de cada uno de los vectores columna, que corresponden finalmente a la salida de la función.

```
for i=1:(round(length(ElSauce)/30)-1)
        DDAmensual = 0;
        TmaxMensual = 0;
        TminMensual = 0;
for j=1:30
DDAmensual=DDAmensual+ElSauce((i-1)*30+j)+Bellavista((i-1)*30+j)+Miraflores((i-1)*30+j);
TmaxMensual = TmaxMensual + Tmax((i-1)*30+j);
TminMensual = TminMensual + Tmin((i-1)*30+j);
end
    LosAndesDDA(i) = DDAmensual;
    LosAndesTmax(i) = TmaxMensual;
    LosAndesTmin(i) = TminMensual;
end
end % if

Promedio(1) = mean2(LosAndesDDA);
Promedio(2) = mean2(LosAndesTmax);
Promedio(3) = mean2(LosAndesTmin);

LosAndesDDA = LosAndesDDA./Promedio(1);

LosAndesTmax = LosAndesTmax./Promedio(2);

LosAndesTmin = LosAndesTmin./Promedio(3);
```

Figura 4.3 Obtención de datos mensuales y normalización

4.1.1.7.2. Red dinámica y estática

Como se explica en 4.1.1.5.1, se tienen dos opciones para configurar la arquitectura de la red, según la forma en que se le entreguen los datos para su entrenamiento. La primera opción es el diseño de una red propiamente dinámica, con el uso de retardos en la capa de entrada. La segunda opción, es el diseño manual de una red dinámica, que en el fondo se trata de una red estática, pero entrenada con datos desfasados, es decir, se arma el set de ejemplos para su entrenamiento, de tal forma que siempre la red recuerde valores pasados, como lo hace una red dinámica.

Para llevar a cabo lo anterior, es necesario saber implementar una red dinámica y estática en Matlab. Para ambos casos hay que tener ciertas consideraciones.

Para el caso de una red dinámica, el vector de entrada y salida deben ser *cell arrays*, es decir, un arreglo de celdas. Para definir un arreglo de celdas se deben representar sus elementos entre llaves { }, de lo contrario solo se crea un vector.

Adicionalmente, en la definición de la arquitectura de una red dinámica, de debe especificar que el tamaño de la entrada es 1, ya que el tamaño de la primera capa, lo define la cantidad de retrasos representada mediante la matriz de retardos.

Para el caso de una red estática, el vector de entrada y salida deben ser simplemente vectores columna. Para definir un vector se deben representar sus elementos entre paréntesis ( ). Adicionalmente, en la definición de la arquitectura de una red estática, el tamaño de la entrada corresponde a la cantidad de filas de la matriz de entrada, el cual, define finalmente el tamaño de la primera capa de la red.

En la Figura 4.4, se observa la definición de una red dinámica seguida de la definición de una red estática, en la que el tamaño de las entradas están descritas con el comando *net.input{1}.size* según se explica en los párrafos anteriores.

```
elseif (argNetKind == 'D' & argLinearity == 'N')
    net = newfftd(In, Out, [1], hide, {'logsig','purelin'});

    net.numInputs = 1;
    net.inputs{1}.size = 1;
    net.inputConnect = [1;0];
    net.biasConnect = [0;0];
    net.layerConnect = [0 0;1 0];
    net.outputConnect = [0 1];
    net.inputWeights{1,1}.delays = delayDDA;

elseif (argNetKind == 'S' & argLinearity == 'N')
    net = newfftd(In, Out, [0], hide, {'logsig','purelin'});

    net.numInputs = 1;
    net.inputs{1}.size = in;
    net.inputConnect = [1;0];
    net.biasConnect = [0;0];
    net.layerConnect = [0 0;1 0];
    net.outputConnect = [0 1];
```

Figura 4.4 Definición de redes dinámicas y estáticas no lineales

4.1.1.7.3. Definición de una red.

Aprovechando la Figura 4.4, se pueden explicar todos los comandos de la definición de una red neuronal con el comando o función *newfftd* (comando utilizado para crear todas las redes neuronales del proyecto). Primeramente, se debe dar un nombre a la red neuronal, que, en el caso de la imagen, ambas redes se llaman *net* (no es posible que dos redes tengan el mismo nombre, pero en la imagen, ambas definiciones no ocurren simultáneamente porque son parte de una función condicional *if*).

En el argumento de la función que crea la red, se entregan las matrices de entrada y salida para su entrenamiento, correspondiente a las variables '*In'* y '*Out'* en la imagen. Posteriormente, en el argumento de la función se entrega una matriz que identifica la presencia o ausencia de retardos de entrada, siendo *'1'* para red dinámica y *'0'* para red estática. Luego, se ingresa al argumento de la función la matriz o variable que indique el tamaño de las capas interiores *('hide')*, seguido de las funciones de activación por cada capa de la red.

Posterior a la primera línea de comando en la Figura 4.4, le siguen especificaciones de su arquitectura. Las líneas *net.numInputs* y *net.input{1}.size* corresponden a la cantidad de entradas y al tamaño de la primera entrada (definido con la variable *'in'* en el caso de la red estática) respectivamente. Las siguientes líneas de comando están definidas con matrices que especifican todas las conexiones de entradas, valores de sesgo y capas, a cada una de las capas de la red, *net.inputConnect*, *net.biasConnect*, y *net.layerConnect* respectivamente. Cada fila *i* corresponde a las conexiones ligadas a cada capa *i*, y cada columna *j*, en el caso de las conexiones entre capas (*net.layerConnect*), corresponde a la unión de la salida de la capa *j* a la entrada de la capa *i*.

Finalmente, en el caso de la red dinámica, existe una séptima línea de comando, *net.inputWeights{1,1}.delays* que representa el vector de *delays*, que contiene los

retardos para cada una de las neuronas en la entrada, y como se mencionó antes, define el tamaño de la capa de entrada de la red.

Otro aspecto importante de mencionar, es la definición del algoritmo de entrenamiento a utilizar. Esta especificación debiese estar en el argumento de la función que define la red, pero como se ve en la Figura 4.4, a continuación de las funciones de activación no se especifica nada más. Esto se debe a que el algoritmo de entrenamiento por defecto corresponde al llamado *Levenberg Marquardt*, que es exactamente, el algoritmo backpropagation que se escoge en la sección 2.2, como mejor alternativa de entrenamiento.

4.1.1.7.4. Linealidad de la red

Para determinar si una red es lineal, o no, se debe tomar en cuenta si las funciones de activación de las capas intermedias de la red son lineales. En general una

```
if (argNetKind == 'D' & argLinearity == 'L')
    net = newfftd(In, Out, [1]);

    net.numInputs = 1;
    net.inputs{1}.size = 1;
    net.inputConnect = [1];
    net.biasConnect = [0];
    net.layerConnect = [0];
    net.outputConnect = [1];
    net.inputWeights{1,1}.delays = delayDDA;

elseif (argNetKind == 'S' & argLinearity == 'L')
    net = newfftd(In, Out, [0]);

    net.numInputs = 1;
    net.inputs{1}.size = in;
    net.inputConnect = [1];
    net.biasConnect = [0];
    net.layerConnect = [0];
    net.outputConnect = [1];
```

Figura 4.5 Definición de redes dinámicas y estáticas lineales

función de activación lineal en la capa de salida de la red, no define la linealidad de la red completa. En el caso en que la red no es lineal, la función de activación lineal en la última capa solo cumple la función de aumentar el rango del valor numérico de la salida (y no limitar su valor como lo hacen las funciones sigmoidales). Por lo tanto, en el caso de una red neuronal lineal, todas sus funciones de activación son lineales.

Para definir una red lineal en la que, en teoría, todas sus capas tienen funciones de activación lineales, no es necesario que tenga más de una capa en su arquitectura, ya que al tener más de una capa "lineal" estas se vuelven redundantes. Siguiendo esto y tal como lo muestra la Figura 4.5, ambas redes, dinámica y estática, están compuestas por solo una capa.

Como se menciona en la sección 4.1.1.7.3 y se observa en la Figura 4.4, la implementación de las funciones de activación se lleva a cabo mediante la asignación, capa por capa, en el argumento de la función que define una red neuronal. En la imagen, ambas redes no son lineales, ya que en su capa intermedia se asigna la función de activación no lineal *logsig*, que corresponde a la función logarítmica sigmoidal.

En la Figura 4.5, no es necesario especificar en el argumento de la función que define la red, una función de activación lineal, ya que, por defecto le asigna a la red la función *purelin*, la cual corresponde a una función lineal.

4.1.1.7.5. Incorporación de una nueva variable

Como se explica en 4.1.1.5.3, se considera la opción de incorporar una nueva variable de entrada correspondiente a la temperatura. Se quiere implementar 3 opciones de redes neuronales, una, considerando solamente la demanda diaria de agua, otra, considerando la demanda y temperaturas diarias máximas, y, por último, una red que considere la demanda, temperaturas máximas y temperaturas diarias mínimas.

La Figura 4.4, nos muestra las definiciones de una red dinámica y otra estática que solo consideran la demanda diaria de agua, como bien lo especifica la línea de comando *net.numInputs = 1;* donde el número de entradas es 1.

```
elseif (argNetKind == 'D' & argLinearity == 'N')
    net = newfftd(In, Out, [1], hide, {'logsig','purelin'});

    net.numInputs = 2;
    net.inputs{1}.size = 1;
    net.inputs{2}.size = 1;
    net.inputConnect = [1 1;0 0];
    net.biasConnect = [0;0];
    net.layerConnect = [0 0;1 0];
    net.outputConnect = [0 1];
    net.inputWeights{1,1}.delays = delayDDA;
    net.inputWeights{1,2}.delays = delayTemp;

elseif (argNetKind == 'S' & argLinearity == 'N')
    net = newfftd(In, Out, [0], hide, {'logsig','purelin'});

    net.numInputs = 2;
    net.inputs{1}.size = in;
    net.inputs{2}.size = 3;
    net.inputConnect = [1 1;0 0];
    net.biasConnect = [0;0];
    net.layerConnect = [0 0;1 0];
    net.outputConnect = [0 1];
```

Figura 4.6 Definición de redes dinámica y estática incluyendo temperaturas máximas

La Figura 4.6 muestra cómo varía la definición de las redes dinámicas y estáticas cuando incorporamos la nueva variable, temperatura máxima. La línea que indica la cantidad de entradas en la red ha cambiado a 2, *net.numInputs = 2*, y aparece una nueva que nos indica el tamaño de la segunda entrada *net.inputs{2}.size = 3*. De la misma forma, el tamaño de la primera entrada, que representa los datos de demanda, es representado por la variable *'in', net.inputs{1}.size = in.*

Para el caso de la red dinámica, es necesario definir el vector de retardos correspondiente a las temperaturas máximas, lo cual se ve en la línea

net.inputWeights{1,2}.delays = delayTemp, que estrictamente corresponde a la asignación de retardos de la matriz de pesos sinápticos de la capa 1 y de la entrada 2. El vector de retardos de temperaturas, junto al vector de retardos de demanda de agua, representan finalmente el tamaño total de la capa de entrada de la red.

```
if (argNetKind == 'D' & argLinearity == 'L')
    net = newfftd(In, Out, [1]);

    net.numInputs = 3;
    net.inputs{1}.size = 1;
    net.inputs{2}.size = 1;
    net.inputs{3}.size = 1;
    net.inputConnect = [1 1 1];
    net.biasConnect = [0];
    net.layerConnect = [0];
    net.outputConnect = [1];
    net.inputWeights{1,1}.delays = delayDDA;
    net.inputWeights{1,2}.delays = delayTemp;
    net.inputWeights{1,3}.delays = delayTemp;

elseif (argNetKind == 'S' & argLinearity == 'L')
    net = newfftd(In, Out, [0]);

    net.numInputs = 3;
    net.inputs{1}.size = in;
    net.inputs{2}.size = 3;
    net.inputs{3}.size = 3;
    net.inputConnect = [1 1 1];
    net.biasConnect = [0];
    net.layerConnect = [0];
    net.outputConnect = [1];
```

Figura 4.7 Definición de redes dinámica y estática incluyendo temperaturas máximas y mínimas

La extracción del código en la Figura 4.7 nos muestra cómo cambia la definición de las redes dinámicas y estáticas incorporando la variable que representa las temperaturas mínimas. Al igual que el caso anterior, la línea que indica la cantidad de entradas en la red ha cambiado a 3, *net.numInputs = 3*, y aparece una nueva línea que nos indica el tamaño de la tercera entrada *net.inputs{3}.size = 3*.

Nuevamente, para el caso de la red dinámica, se define el vector de retardos correspondiente a las temperaturas mínimas, lo cual se ve reflejado en la línea *net.inputWeights{1,3}.delays = delayTemp*. El vector de retardos de temperaturas mínimas, máximas, y de demanda de agua, en conjunto, representan el tamaño total de la capa de entrada de la red.

4.1.1.7.6. Entrenamiento y simulación de la red

Para llevar a cabo la implementación del entrenamiento de la red neuronal, es preciso seguir algunos pasos y definir ciertos parámetros.

```
%%%% TRAIN

 % Initialization
    net = init(net);

    % Train
    ts=.1;
    epoca = 100;
    error = 1e-10;
    lr = 0.01;
    gradiente = 0;
    mu = 1;
    mu_dec = 0.8;
    mu_inc = 1.2;

    % Training parameters
    net.trainParam.epochs = epoca;
    net.trainParam.goal = error;
    net.trainParam.min_grad = gradiente;
    net.trainParam.alpha = lr;
    net.trainParam.mu = mu;
    net.trainParam.mu_dec = mu_dec;
    net.trainParam.mu_inc = mu_inc;

    net = train(net, In, Out);
```

Figura 4.8 Entrenamiento de una red neuronal

La extracción del código en la Figura 4.8 nos muestra los pasos a seguir para entrenar una red neuronal. Primeramente, se deben inicializar aleatoriamente los valores de las matrices que contienen los pesos sinápticos de la red, lo cual se lleva a cabo con la línea *net = init(net)*. Cada vez que se ejecuta el código completo, y la red vuelve a ser creada, entrenada y simulada, estas matrices de pesos sinápticos, se inicializan con nuevos valores.

Luego se procede a definir parámetros de entrenamiento tal como aparece en la imagen. Algunos de los parámetros de la figura anterior se pueden familiarizar con el análisis del entrenamiento backpropagation explicado en la sección 3.3.4.1, como por ejemplo, la cantidad de épocas máximas con las que la red repite el algoritmo de entrenamiento y la razón de aprendizaje con la que el algoritmo alcanza el valor óptimo para los pesos sinápticos. También se definen el mínimo gradiente y el error meta del entrenamiento.

Uno de los parámetros restantes en la imagen, la variable *mu*, corresponde al valor inicial de μ de la ecuación recursiva 2.2 expuesta en la sección 2.1.3, que define la actualización de los pesos sinápticos de la red durante su entrenamiento. Este valor es multiplicado por el parámetro *mu_dec* cada vez que la función de desempeño (función que cuantifica el error que va cometiendo la red durante su entrenamiento) es reducida en una iteración, y es multiplicada por *mu_inc* cada vez que la función de desempeño se incrementa [3].

En general, el algoritmo de entrenamiento utilizado en este trabajo, Levenberg Marquardt, es un algoritmo rápido, es decir, que converge rápidamente. Ante esto, es necesario definir los parámetros de entrenamiento de tal forma que la convergencia sea relativamente lenta. Para lograr esto, según [8], el parámetro mu debe valer 1, *net.trainParam.mu = 1*, y los valores de mu_dec y mu_inc deben tomar valores cercanos a 1, como se muestra en la Figura 4.8, *net.trainParam.mu_dec = 0.8*, y *net.trainParam.mu_inc = 1.2*.

Por último, para entrenar la red, se utiliza la función *train*, como se ve en la Figura 4.8 en la línea *net = train(net, In, Out)*. Este comando tiene como argumentos la red *net*, y el set de ejemplos de entradas y salidas representadas por las matrices *In* y *Out*, definidas previamente en el código.

4.1.2. Filtración de datos

En esta sección se da una alternativa de mejora en la predicción de los datos mediante la filtración de estos, es decir, quitándole la aleatoriedad a los datos de demanda de agua, y conservando así, la conducta generalizada del consumo de agua en los clientes.

Para filtrar los datos, se crea una función de filtraje que pide como argumentos los datos originales, y un valor *n*, tal como lo muestra la Figura 4.9.

```
function[LosAndesDDA_F,LosAndesTmax_F,LosAndesTmin_F]=Filter_n(argDDA,argTmax,argTmin,n)

for i=1:(length(argDDA)-n+1)

    SLosAndesDDA_F = 0;
    SLosAndesTmax_F = 0;
    SLosAndesTmin_F = 0;

    for j=1:n
    SLosAndesDDA_F = SLosAndesDDA_F + argDDA(i+j-1);
    SLosAndesTmax_F = SLosAndesTmax_F + argTmax(i+j-1);
    SLosAndesTmin_F = SLosAndesTmin_F + argTmin(i+j-1);
    end

    LosAndesDDA_F(i) = SLosAndesDDA_F/n;
    LosAndesTmax_F(i) = SLosAndesTmax_F/n;
    LosAndesTmin_F(i) = SLosAndesTmin_F/n;
end
```

Figura 4.9 Filtración de datos

La Figura 4.9 representa un extracto del código de la función que entrega los datos de demandas y temperaturas filtrados mediante el promedio móvil de datos consecutivos. El argumento de la función, *n*, representa la cantidad de datos consecutivos que se utilizan para generar los promedios.

Para obtener los promedios, es necesario generar variables inicializadas en cero, que correspondan a la suma de *n* datos consecutivos para su posterior división por *n*.

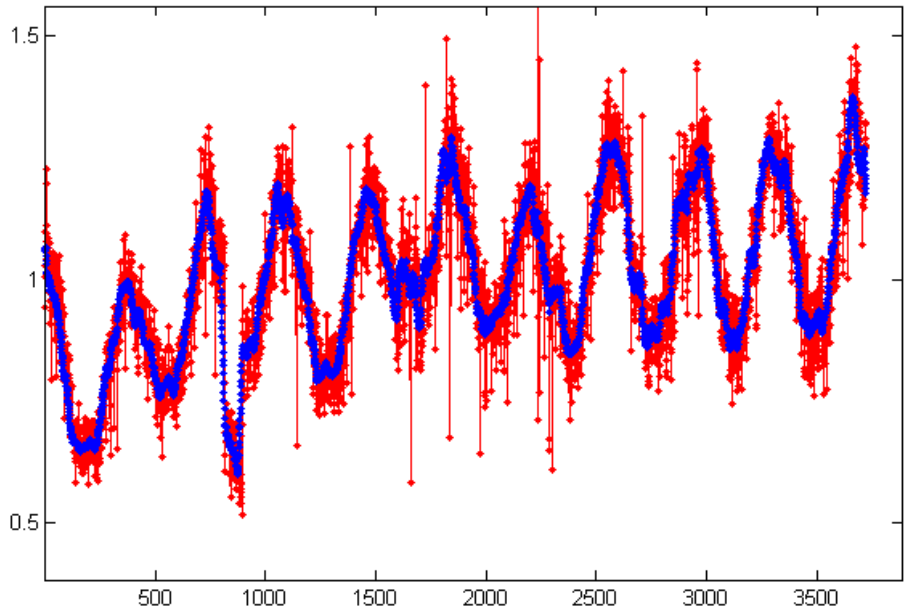


Figura 4.10 Datos diarios reales y filtrados de demanda de agua

Esta acción es repetida para cada set de datos, obteniendo nuevas matrices de datos para entrenar y simular nuevas redes neuronales.

La Figura 4.10 nos muestra una gráfica de los datos reales de demanda de agua, en rojo, versus los datos filtrados, en azul. Se observa que los datos reales tienen un comportamiento estacional según la temporada del año, recordando que esta gráfica corresponde a un tiempo total de 10 años aproximadamente. En la gráfica, se ve que el comportamiento cíclico de los datos reales se mantiene en los datos filtrados. A continuación, en la Figura 4.11, se hace un acercamiento de la gráfica, para mostrar la efectividad del filtro diseñado.

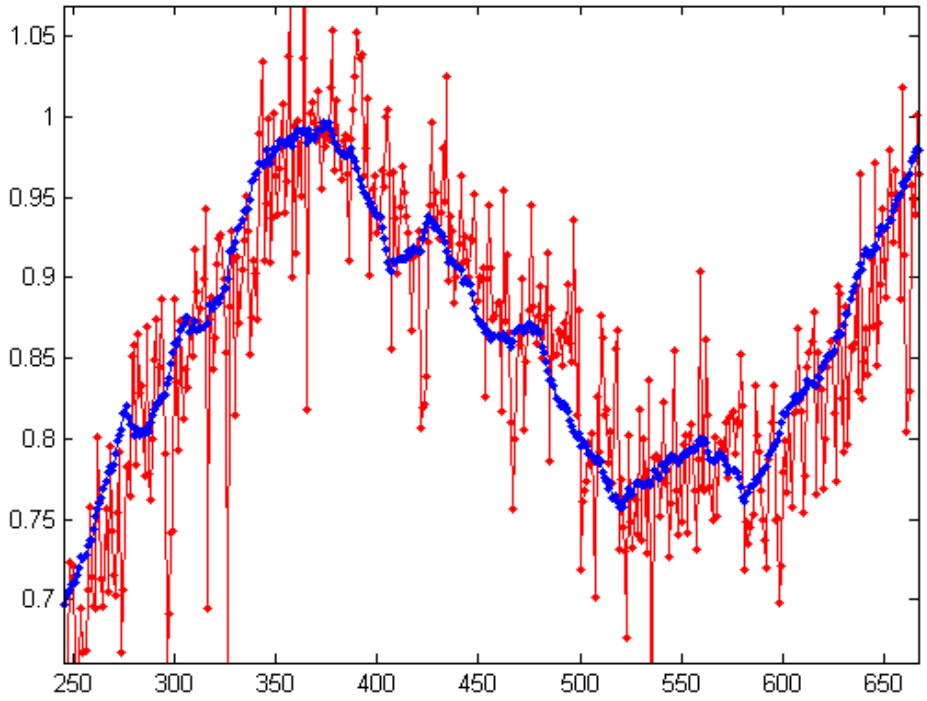


Figura 4.11 Datos filtrados de demanda de agua

El filtraje de datos de demanda de la Figura 4.11, en azul, fue obtenido mediante el promedio móvil de 20 datos consecutivos por cada dato filtrado, es decir *n=20*. Lo mismo se realiza, para los datos de temperaturas máximas diarias y temperaturas mínimas diarias, en el caso de que se decida utilizar estas variables.

En el caso de los datos mensuales, se considera que no es necesario un filtraje de sus valores, ya que, como se muestra en la Figura 4.12, los datos mensuales de demanda de agua representan de buena manera, y sin mucha desviación, el comportamiento de consumo de los clientes.

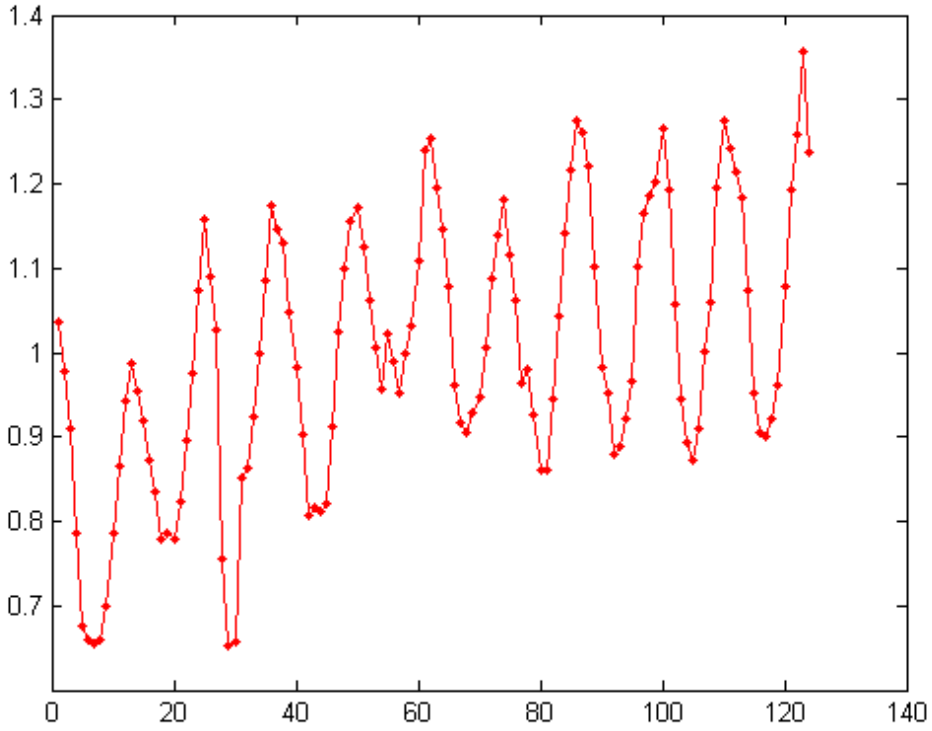


Figura 4.12 Datos mensuales de demanda de agua

4.1.3. Re-entrenamiento

En esta sección se explica una última propuesta de mejora para el pronóstico de agua potable. Esta mejora consiste en re-entrenar las redes ya seleccionadas para las predicciones. Esto quiere decir, que la inicialización aleatoria de pesos sinápticos requerida para el entrenamiento de la red, ya no es necesaria, ya que los valores iniciales de los pesos sinápticos son los alcanzados en la primera etapa de entrenamiento.

Para re-entrenar una red, se construye una función que pide como argumentos, los datos de agua y temperaturas, especificación de la frecuencia de datos (diaria o mensual), y el desfase de entrenamiento (concepto explicado en 4.1.1.5.5).

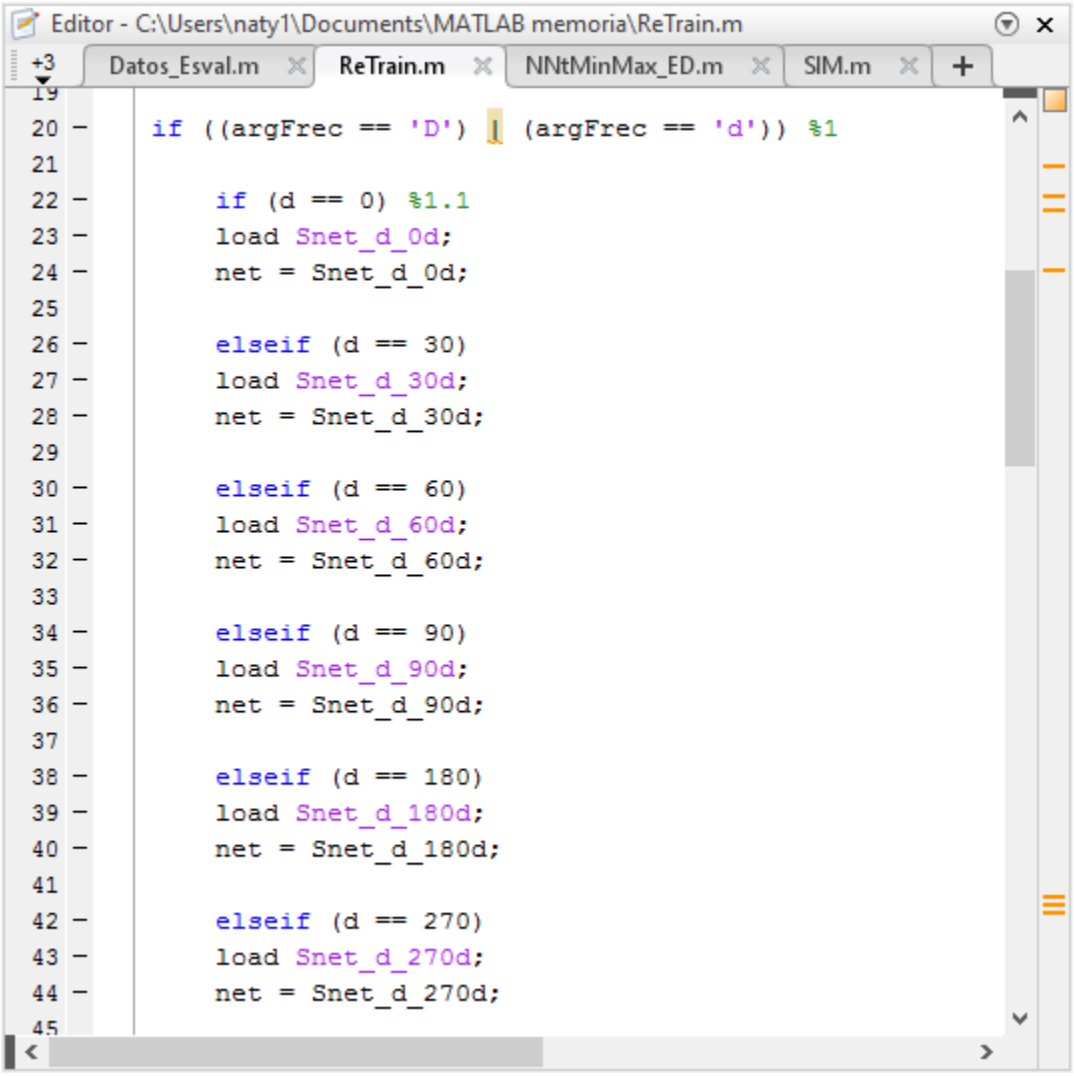


Figura 4.13 Selección de red neuronal para su re-entrenamiento

La Figura 4.13, muestra un extracto del código de la función que re-entrena las redes. Según la especificación de la frecuencia de los datos y el desfase de entrenamiento, el código carga la red correspondiente a esa información, para luego re-entrenarla. En la figura se observa la sección del código en la que la función corre si la frecuencia de los datos es diaria, *(argFrec == 'D') | (argFrec == 'd')* para

posteriormente asignar a la variable *net* la carga de la red correspondiente al desfase de entrenamiento ingresado, el cual puede variar desde los 0 *(d == '0')* a los 360 *(d == '360')* días de desfase.

Posteriormente, la red cargada es entrenada con los mismos parámetros de entrenamiento especificados en 4.1.1.7.6, para luego arrojar el error cometido por la red y comparar resultados.

CAPÍTULO 5

Resultados

A continuación, se presentan los resultados de todos los puntos especificados en la metodología de trabajo, ya sea, en el diseño de la red, entrenamiento, simulación y todas las decisiones tomadas según las opciones presentadas en el capítulo anterior.

Posteriormente se presentan las conclusiones del proyecto, incluyendo posibles mejoras y propuestas.

5.1. Resultados

5.1.1. Decisiones de diseño

Como se explica en la sección 4.1.1.5, se quiere analizar varias aristas del diseño de la red a utilizar para el pronóstico de demanda de agua potable. Entre las opciones a considerar se encuentran el uso de temperaturas mínimas y máximas, la elección de la linealidad de la red, y la utilización de una red estática o dinámica.

La metodología de análisis de estas opciones consiste en elegir la mejor opción entre redes lineales o no lineales, y redes estáticas o dinámicas, para, posteriormente, utilizar la configuración de red escogida para evaluar la incorporación de la temperatura.

5.1.1.1. Tipo de red y linealidad

Para tomar la primera decisión, se programan 4 redes neuronales, que tengan como entrada solamente la información de demanda de agua diaria, correspondiente a las siguientes combinaciones:

- Red dinámica lineal.
- Red dinámica no lineal.
- Red estática lineal.
- Red estática no lineal.

A continuación, se procede a mostrar las gráficas de resultados para cada una de las opciones. Cada caso es diseñado en base a una arquitectura de 3 capas, donde la capa de entrada, oculta y de salida tienen las dimensiones *12/9/1* respectivamente. Las redes escogidas para tomar esta decisión fueron obtenidas mediante una repetición sucesiva de entrenamientos, para finalmente, escoger la red de mejor desempeño.

A continuación, las imágenes muestran, (en rojo), la demanda de agua diaria real versus, la demanda de agua diaria pronosticada, (en azul), de dichas redes seleccionadas para cada caso.

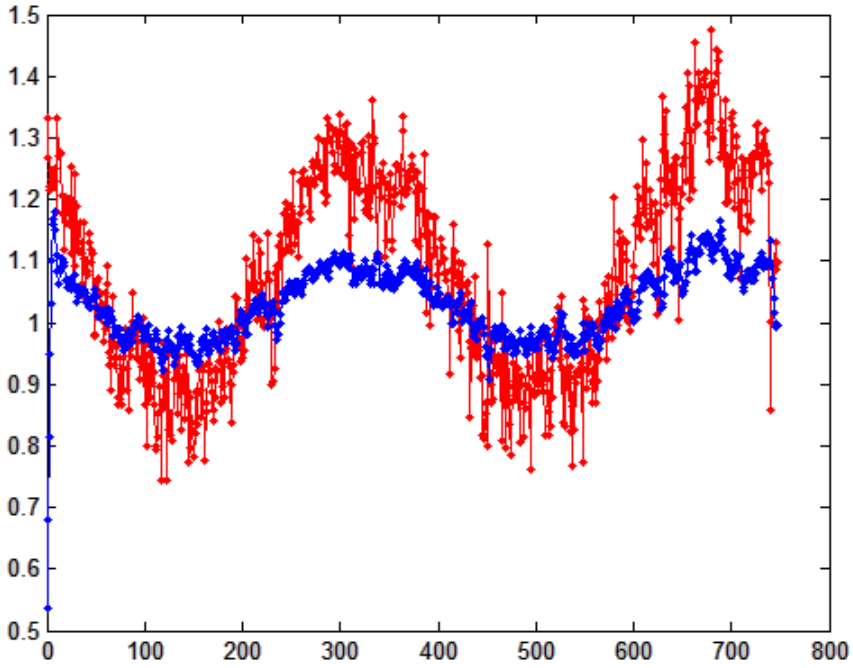


Figura 5.1 Simulación de red dinámica lineal

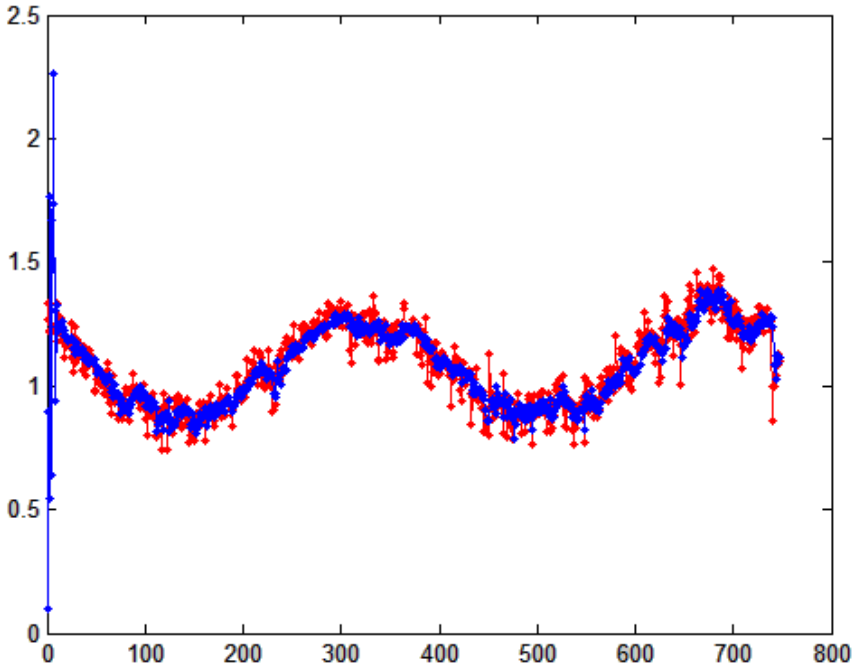


Figura 5.2 Simulación de red dinámica no lineal

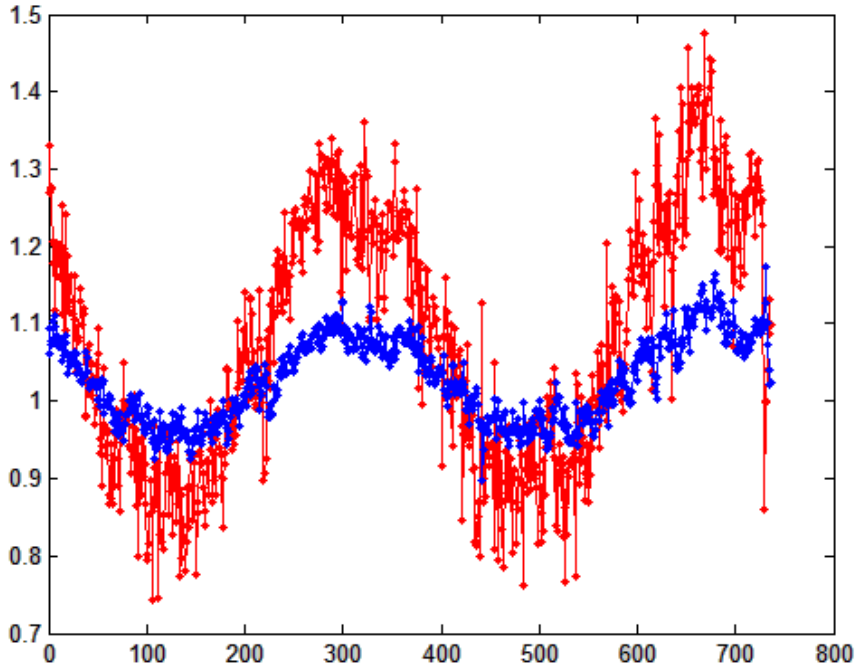


Figura 5.3 Simulación de red estática lineal

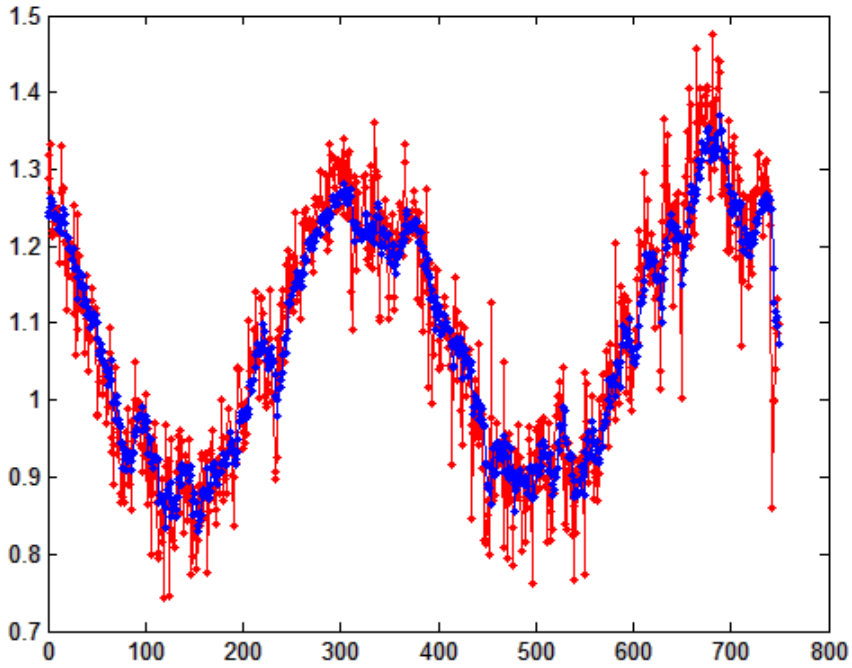


Figura 5.4 Simulación de red estática no lineal

Los resultados según los parámetros de evaluación, *MSE*, *MAPE* y *MSPE*, explicados en 4.1.1.6, son expuestos en la Tabla 5.1.

	DINÁMICA LINEAL	DINÁMICA NO LINEAL	ESTÁTICA LINEAL	ESTÁTICA NO LINEAL
MSE	0.0174	0.0100	0.0167	0.0041
MAPE (%)	9.4789	5.3606	9.5124	4.6800
MSPE (%)	1.3032	0.7804	1.2735	0.3972

Tabla 5.1 Resultados de simulaciones según redes estáticas o dinámicas, y linealidad

Como se puede observar en la Figura 5.1 y Figura 5.3, definitivamente la opción de utilizar redes neuronales lineales no es acertada. Como se menciona en la sección 4.1.1.5.2, la linealidad en los datos no era esperada, ya que estos representan el consumo de agua potable en personas, por lo tanto, no hay una regularidad en el comportamiento de los datos. Adicionalmente, como se ve en la Tabla 5.1, los errores

de las redes lineales son considerablemente mayores que los errores de redes no lineales. Por lo tanto, como primera decisión, se elige utilizar redes del tipo no lineales.

Respecto a elegir entre una red estática o una red dinámica, se puede ver en la Figura 5.2 y Figura 5.4 que los pronósticos son bastante acertados y parecidos entre sí. Observando los errores en la Tabla 5.1, se ve que sus valores son bastante parecidos, lo cual tiene sentido, ya que como se explica en la sección 4.1.1.5.1, ambas redes reciben los mismos sets de ejemplos para su entrenamiento, solo que, en diferente forma, por lo tanto, ambas alternativas son factibles de utilizar.

Para tomar una decisión, ante estas dos opciones, se toma en consideración que los primeros sets de ejemplos en las redes dinámicas están errados, esto ocurre *in* veces, siendo *in* el tamaño de la primera capa. Esto se explica analizando cómo la red va aplicando los retardos a los primeros datos de entrada. Por ejemplo, al ingresar por primera vez el vector de datos, la entrada que tiene retardo cero lee el primer dato, y el resto de las entradas (con retardos distintos de 0) leen ceros, ya que aún no existen valores pasados. En el segundo ingreso de datos, la entrada que tiene retardo cero lee el segundo dato del vector de datos, la entrada con retardo 1 lee el primer dato del vector, y el resto de las entradas leen ceros. Esto ocurre sucesivamente hasta que la entrada que tiene la mayor cantidad de retardos pueda recordar un dato pasado distinto de 0. Adicionalmente, el error de la red dinámica no lineal según la Tabla 5.1 es levemente mayor que el error de la red estática no lineal, por lo tanto, basado en estos dos argumentos, se decide utilizar la red estática no lineal.

5.1.1.2. Temperaturas

Para evaluar la opción de agregar datos de temperatura al entrenamiento de la red, según la sección 4.1.1.5.3, se diseña una red neuronal para cada desfase de entrenamiento y con 3 configuraciones de entrada diferentes, una incorporando solamente la demanda de agua, otra agregando la variable de temperaturas máximas y

una última configuración utilizando todos los datos disponibles (demanda de agua, temperatura máxima y mínima). Para cada caso, se procede a medir el error porcentual absoluto medio entre la predicción realizada por la red y los datos reales.

5.1.1.2.1. Caso diario

Para cada configuración, en el caso de redes que trabajan con datos diarios, se agrega a la capa de entrada 3, 6 y 12 neuronas por variable de temperatura adicional. Es importante tener claro que el número total de entradas de la arquitectura de cada alternativa representa la suma de 50 neuronas de demanda de agua más las 3, 6 o 12 neuronas de entrada de cada variable de temperatura adicional. El número de neuronas ocultas de la capa intermedia es 30 para todos los casos.

A continuación, la Tabla 5.2 presenta los errores absolutos porcentuales para cada una de las opciones mencionadas.

	DESFASE	**MAPE (%)**	**MAPE (%)**	**MAPE (%)**	**MAPE (%)**
NEURONAS POR VARIABLE DE TEMP.	-	0	3	6	12
DEMANDA DE AGUA	0	4.7032	-	-	-
	30	6.9809	-	-	-
	60	8.9703	-	-	-
	90	10.0889	-	-	-
	180	10.8323	-	-	-
	270	7.4900	-	-	-
	360	6.1221	-	-	-
ERROR PROMEDIO	**-**	**7.8840**	**-**	**-**	**-**
	0	-	4.7781	4.8392	4.7549
	30	-	6.8828	6.9872	6.9165
	60	-	7.8999	8.1162	8.2780

DEMANDA DE AGUA + TEMP. MÁX.	90	-	8.8956	8.7910	8.5183
	180	-	9.0048	8.6252	7.9705
	270	-	6.9681	7.2588	7.2441
	360	-	6.3962	6.4503	6.6250
ERROR PROMEDIO	-	-	7.2608	7.2954	7.1867
DEMANDA DE AGUA + TEMP. MÁX. + TEMP. MÍN.	0	-	4.8246	4.8358	4.8806
	30	-	6.8460	7.0513	7.0333
	60	-	8.2340	8.1329	8.3137
	90	-	8.4143	8.2219	8.1370
	180	-	8.2437	7.9966	7.9119
	270	-	7.1708	7.4596	7.9678
	360	-	6.2617	6.3778	6.7175
ERROR PROMEDIO	-	-	7.1422	7.1537	7.2802

Tabla 5.2 Resultados de simulaciones según incorporación de la temperatura para el caso diario

Es necesario mencionar, que cada uno de los resultados mostrados en la tabla corresponden a redes neuronales de pesos sinápticos inicializados aleatoriamente, como se menciona en la sección 4.1.1.7.6, seleccionando las redes desde un proceso de entrenamientos y re-entrenamientos, y escogiendo finalmente las de mejor desempeño.

Los resultados de la tabla muestran un claro aumento en el error absoluto de la red al no utilizar las variables de temperatura para el pronóstico, observando que el error se acentúa en redes de desfase de entrenamiento de 90 y 180 días. Para el resto de las opciones no se generan grandes diferencias de desempeño, por lo tanto, la decisión en cuestión se basa en el promedio de los errores de las redes por cada opción. Finalmente, para redes que trabajan con datos diarios, se escoge la configuración de red en la que se utilizan ambas temperaturas con 3 neuronas de entrada por variable.

Para el caso de redes que trabajan con datos diarios filtrados, se estima que la decisión de utilizar las mediciones de temperaturas es la misma que para el caso diario,

ya que se trabaja con redes del mismo tamaño, misma cantidad de datos y mismos desfases de entrenamiento.

5.1.1.2.2. Caso mensual

Para cada opción de configuración, en el caso de redes que trabajan con datos mensuales, se agrega a la capa de entrada 2, 3, 4 y 5 neuronas por variable de temperatura adicional. Es importante tener claro que el número total de entradas de la arquitectura de cada alternativa representa la suma de 12 neuronas de demanda de agua más las 2, 3, 4 o 5 neuronas de entrada de cada variable de temperatura adicional, y, por último, el número de neuronas ocultas de la capa intermedia es 9 para todos los casos.

A continuación, la Tabla 5.3 presenta los errores absolutos porcentuales para cada una de las opciones mencionadas.

	DESFASE	**MAPE (%)**	**MAPE (%)**	**MAPE (%)**	**MAPE (%)**	**MAPE (%)**
NEURONAS POR VARIABLE DE TEMP.	-	0	2	3	4	5
DEMANDA DE AGUA	0	2.8813	-	-	-	-
	1	3.1731	-	-	-	-
	2	3.2362	-	-	-	-
	3	3.5089	-	-	-	-
	6	3.5721	-	-	-	-
	9	3.4841	-	-	-	-
	12	3.7809	-	-	-	-
ERROR PROMEDIO	-	**3.3767**	-	-	-	-
	0	-	2.6534	3.1086	2.6696	2.8908
	1	-	2.9580	3.2604	3.2506	3.5346
	2	-	3.2165	3.4992	3.7324	3.3441

DEMANDA DE AGUA + TEMP. MÁX.	3	-	2.9513	3.4584	3.2113	3.1424
	6	-	3.2396	3.5569	2.9752	3.2194
	9	-	2.5666	2.6129	3.3068	3.3727
	12	-	3.1768	3.8676	3.6316	3.3239
ERROR PROMEDIO	**-**	**-**	**2.9660**	**3.3377**	**3.2539**	**3.2611**
DEMANDA DE AGUA + TEMP. MÁX. + TEMP. MÍN.	0	-	2.5396	3.3283	3.2577	3.6238
	1	-	4.0098	4.2462	4.9833	4.1725
	2	-	3.8359	3.9444	4.2294	4.2566
	3	-	3.3560	3.4955	4.4700	4.0748
	6	-	3.0203	2.9852	2.6484	3.4287
	9	-	1.7759	1.8222	2.1575	2.0666
	12	-	4.7800	3.3642	2.4844	1.6579
ERROR PROMEDIO	**-**	**-**	**3.3311**	**3.3123**	**3.4615**	**3.3258**

Tabla 5.3 Resultados de simulaciones según incorporación de la temperatura para el caso mensual

Al igual que el caso diario, los resultados presentados provienen de redes inicializadas aleatoriamente, y seleccionadas a través de entrenamientos y re-entrenamientos según su desempeño.

En este caso, no hay grandes diferencias entre los errores de predicción de todas las alternativas puestas a prueba. Se asocian sus diferencias a la aleatoriedad de su inicialización, por lo tanto, se procede a escoger la configuración, que, en promedio, tenga el menor error. Finalmente, para redes que trabajan con datos mensuales, se escoge la configuración de red en la que se utilizan la demanda de agua y las temperaturas máximas para la predicción, con un total de 2 neuronas de entrada para la temperatura máxima.

5.1.2. Re-entrenamiento

Como se expone en la sección 4.1.3, las redes son re-entrenadas tomando como inicialización de pesos sinápticos, los mismos pesos de la red entrenada anteriormente.

Teniendo en cuenta las decisiones tomadas en la sección anterior se diseñan redes estáticas no lineales, con distintos desfases de entrenamiento, variables de entrada y frecuencias en los datos. A continuación, las siguientes tablas indican el error porcentual absoluto medio, *MAPE*, para cada una de las redes diseñadas, indicando la cantidad de intentos de re-entrenamiento para llegar al error óptimo. Los errores ilustrados en las tablas siguientes, corresponden al error entre la predicción y los datos reales.

DESFASE (DÍAS)	***0***	***30***	***60***	***90***	***180***	***270***	***360***
MAPE INICIAL	4.8425	6.9300	8.2340	8.4143	8.2437	7.2149	6.2804
MAPE (1ER INTENTO)	4.8302	6.9177	-	-	-	7.1947	6.2779
MAPE (2DO INTENTO)	4.8294	6.8855	-	-	-	7.1942	6.2678
MAPE (3ER INTENTO)	4.8288	6.8498	-	-	-	7.1708	6.2617
MAPE (4TO INTENTO)	4.8246	6.8460	-	-	-	-	-
MAPE ÓPTIMO	***4.8246***	***6.8460***	***8.2340***	***8.4143***	***8.2437***	***7.1708***	***6.2617***

Tabla 5.4 MAPE de redes re-entrenadas con datos diarios

DESFASE (DÍAS)	***0***	***30***	***60***	***90***	***180***	***270***	***360***
MAPE INICIAL	0.2701	3.9880	5.4917	5.7608	5.9801	4.7212	4.4269
MAPE (1ER INTENTO)	0.2688	-	-	-	-	-	-
MAPE ÓPTIMO	***0.2688***	***3.9880***	***5.4917***	***5.7608***	***5.9801***	***4.7212***	***4.4269***

Tabla 5.5 MAPE de redes re-entrenadas con datos diarios filtrados

En este caso, para los resultados de la Tabla 5.5, es importante tener en cuenta que los errores de predicción son obtenidos respecto a los datos reales filtrados. Se observa que los resultados en este caso son más favorables que en el caso de redes entrenadas con datos no filtrados, esto se debe a que de cierta forma se quita el "ruido" representado por la aleatoriedad del consumo de agua diaria, y le genera a la red una disminución en la dificultad del aprendizaje del comportamiento de datos.

DESFASE (DÍAS)	**0**	**1**	**2**	**3**	**6**	**9**	**12**
MAPE INICIAL	2.7633	3.0802	3.2165	3.3707	3.2396	2.6544	3.3053
MAPE (1ER INTENTO)	2.6534	3.0539	-	2.9513	-	2.5680	3.2899
MAPE (2DO INTENTO)	-	3.0379	-	-	-	2.5666	3.2834
MAPE (3ER INTENTO)	-	3.0208	-	-	-	-	3.2471
MAPE (4TO INTENTO)	-	2.9580	-	-	-	-	3.2105
MAPE (5TO INTENTO)	-	-	-	-	-	-	3.1768
MAPE ÓPTIMO	***2.6534***	***2.9580***	***3.2165***	***2.9513***	***3.2396***	***2.5666***	***3.1768***

Tabla 5.6 MAPE de redes re-entrenadas con datos mensuales

Tras el uso de la función de re-entrenamiento, como se ve en las tablas, algunos errores óptimos han sido alcanzados desde el primer re-entrenamiento, y en otros casos, el error va mejorando en cada intento. Cada vez que se ingresa una red con error óptimo a la función de re-entrenamiento, esta arroja el mismo error, por lo tanto, es de esta forma que se entiende que el error óptimo ha sido alcanzado. Finalmente, en negrita, se muestran los errores óptimos alcanzados para cada una de las redes diseñadas.

5.1.3. Resultados diarios

En esta sección se muestran los resultados de simulaciones y errores de las redes neuronales definitivas para el pronóstico de demanda de agua en todas sus

configuraciones. Los errores ilustrados, al igual que en las secciones anteriores, corresponden a los obtenidos entre los pronósticos de cada red y los datos reales según su frecuencia. Para el caso de los datos filtrados, el error es calculado entre los datos pronosticados y los datos diarios filtrados.

Los resultados se presentan según frecuencia y filtración de datos, para cada caso de desfase de entrenamiento.

5.1.3.1. Resultados con datos sin filtrar

A continuación, se representan gráficos de la simulación de cada una de las redes neuronales diseñadas para cada desfase de entrenamiento de datos diarios de demanda de agua y temperaturas mínimas y máximas. Los datos mostrados en las gráficas se encuentran en unidades de litros/día, es decir, la información ya ha sido des-normalizada.

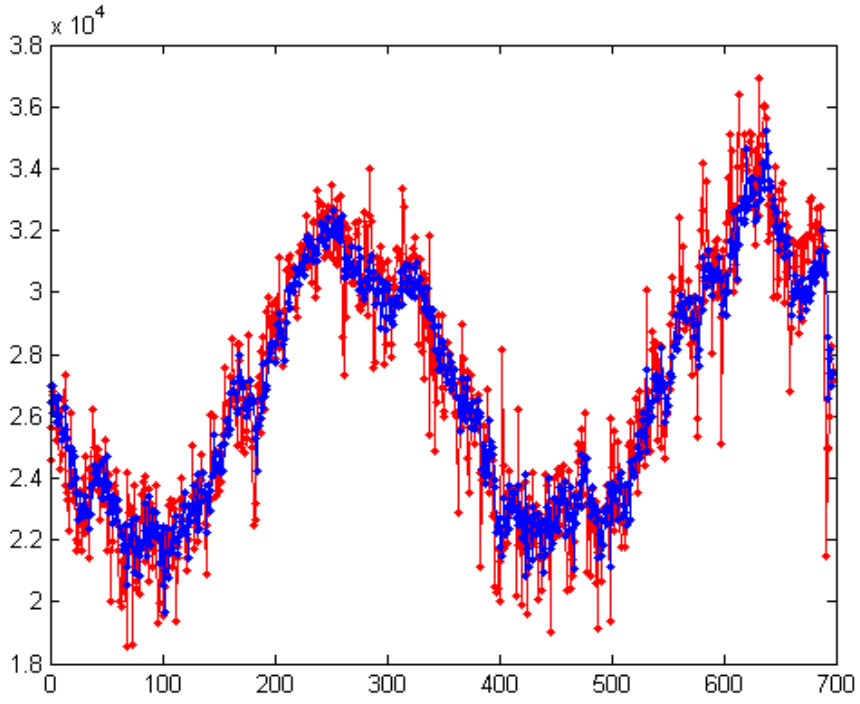


Figura 5.5 Predicción de datos diarios de demanda de agua versus datos reales con desfase de entrenamiento 0

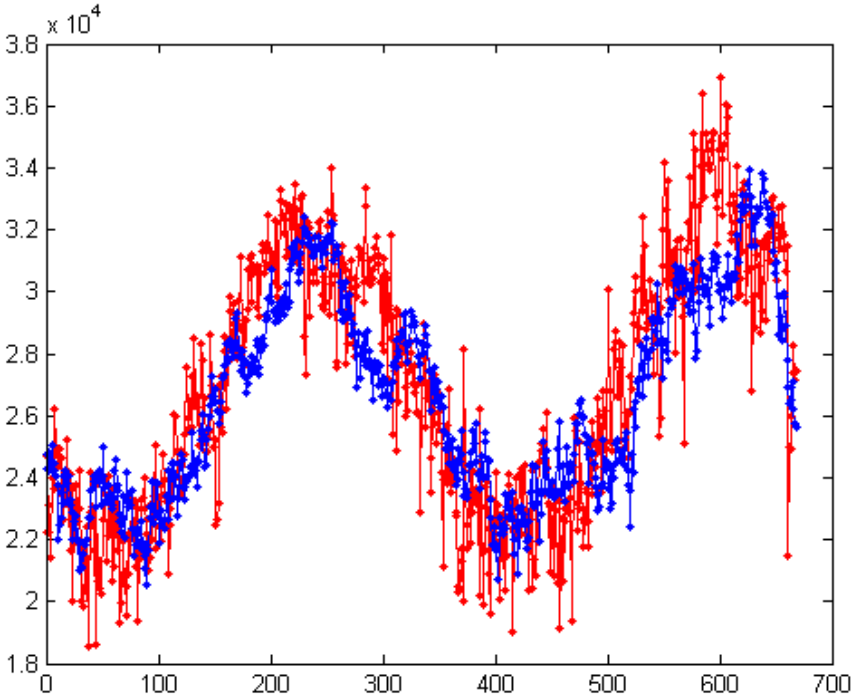


Figura 5.6 Predicción de datos diarios de demanda de agua versus datos reales con desfase de entrenamiento 30 días

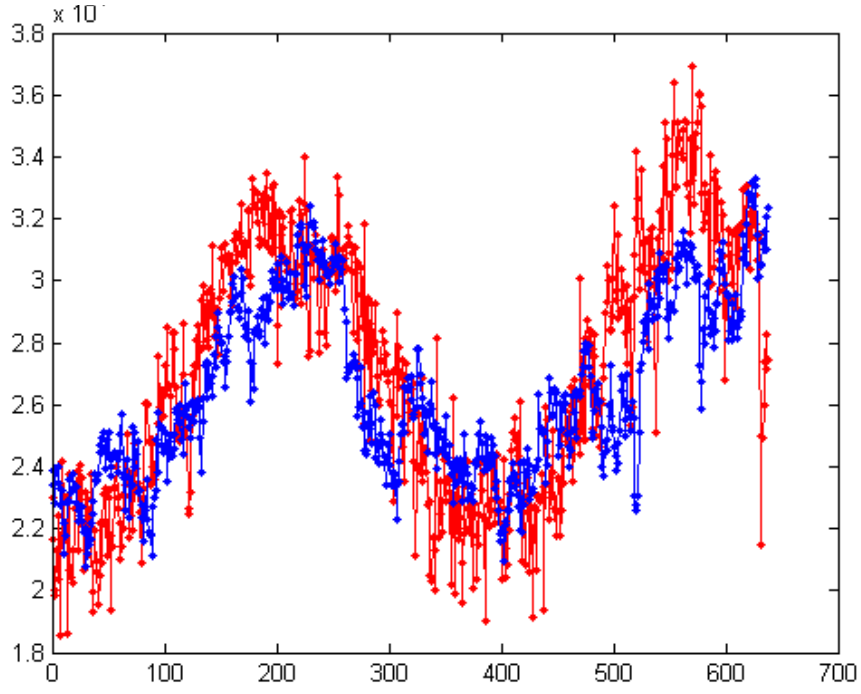


Figura 5.7 Predicción de datos diarios de demanda de agua versus datos reales con desfase de entrenamiento 60 días

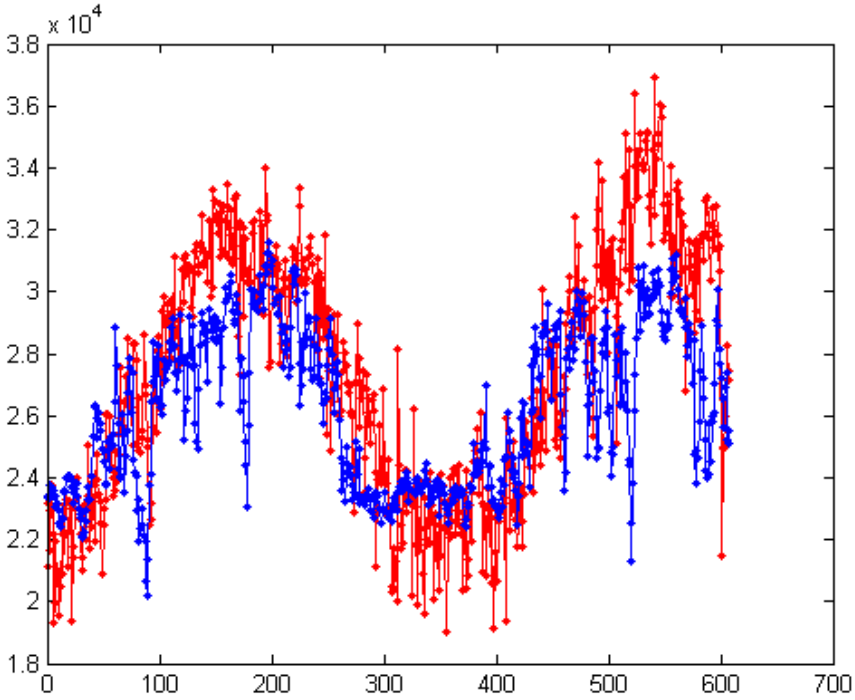


Figura 5.8 Predicción de datos diarios de demanda de agua versus datos reales con desfase de entrenamiento 90 días

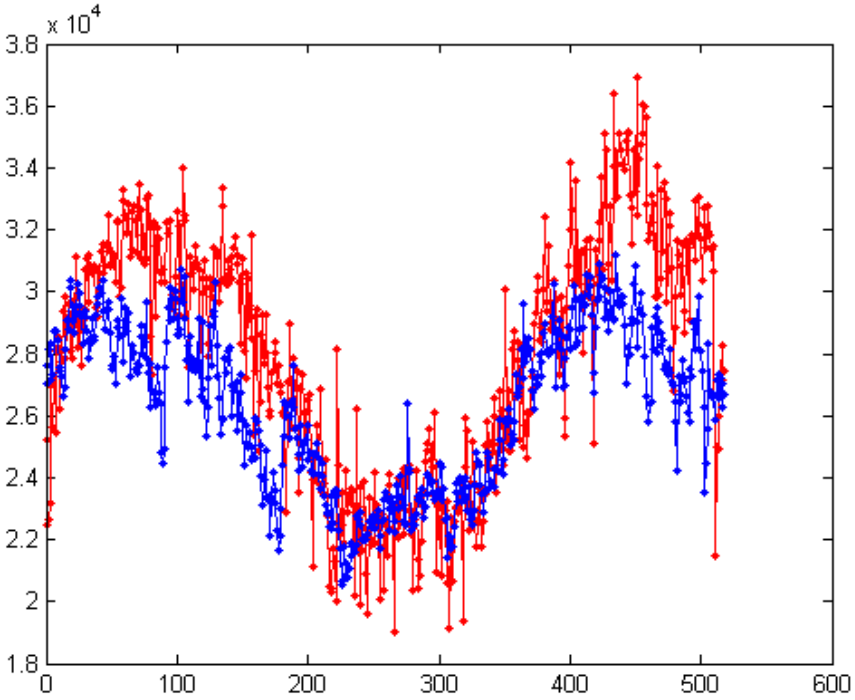


Figura 5.9 Predicción de datos diarios de demanda de agua versus datos reales con desfase de entrenamiento 180 días

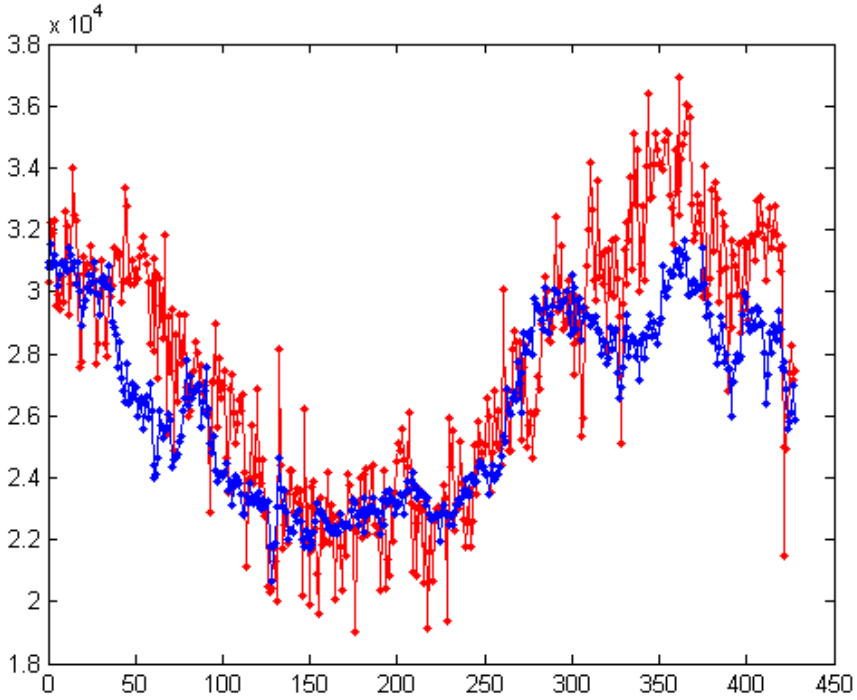


Figura 5.10 Predicción de datos diarios de demanda de agua versus datos reales con desfase de entrenamiento 270 días

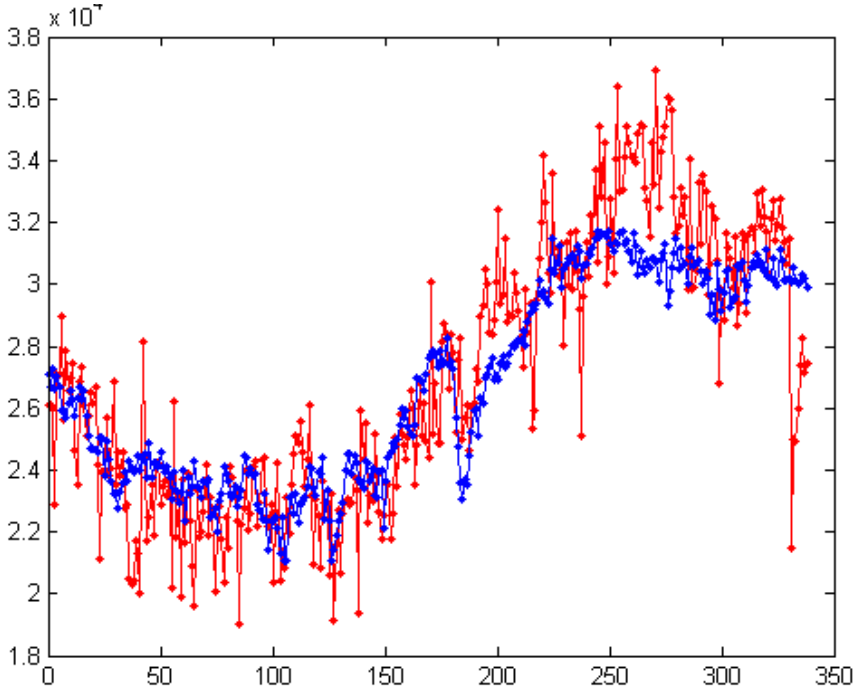


Figura 5.11 Predicción de datos diarios de demanda de agua versus datos reales con desfase de entrenamiento 360 días

De los gráficos se observa que mientras mayor es el desfase de entrenamiento, el pronóstico va perdiendo exactitud hasta un cierto punto, ya que en los últimos gráficos el desempeño de la red vuelve a mejorar. De todas formas, los datos pronosticados, en azul, mantienen la estacionalidad y amplitud de los datos reales, en rojo.

En la Tabla 5.7, se presentan los parámetros de evaluación de cada una de las redes diseñadas según la cantidad de días de desfase en su entrenamiento. Adicionalmente, se muestra la arquitectura de cada una de las redes, especificando la cantidad de neuronas en la capa de entrada, oculta y de salida.

La arquitectura de la red es escogida a través del entrenamiento de redes con distintas arquitecturas, arrojando que la arquitectura *56/30/1* tiene el mejor desempeño. Se debe tener en cuenta que para tomar esta decisión se considera que aumentar aún más el tamaño de la red conlleva una inversión muy grande de tiempo, recordando que una red se entrena y re-entrena muchas veces, y que existen muchas configuraciones de redes a diseñar.

DESFASE (DÍAS)	MSE	MAPE (%)	MSPE (%)	ARQUITECTURA
0	0.0042	4.8246	0.4080	56/30/1
30	0.0083	6.8460	0.7240	56/30/1
60	0.0125	8.2340	1.0563	56/30/1
90	0.0147	8.4143	1.0952	56/30/1
180	0.0148	8.2437	1.0284	56/30/1
270	0.0109	7.1708	0.7859	56/30/1
360	0.0074	6.2617	0.6579	56/30/1

Tabla 5.7 Errores y arquitectura de cada red entrenada con datos diarios sin filtrar

Mientras mayor es el desfase de entrenamiento, mayor es el error en los pronósticos hasta un cierto desfase, ya que, desde los 180 días de desfase la red vuelve

a mejorar su pronóstico. Para la primera mitad de los casos, en que la red va aumentando su error de predicción, se entiende que la red tiene una mayor dificultad en el aprendizaje de los datos debido al aumento en el desfase de entrenamiento, hasta que luego de los 180 días, es decir, 6 meses, (recordando la forma cíclica de los datos), para la red se vuelve más sencillo captar el comportamiento de la red, ya que, en este caso, los datos de entrada tienen un desfase de medio periodo respecto a los datos de salida. Así mismo se puede entender la mejora en el desempeño de la red cuando ha pasado un periodo completo, en el caso del desfase de 360 días.

5.1.3.2. Resultados con datos filtrados

En esta sección se presentan los resultados obtenidos para el pronóstico diario de agua potable con datos filtrados según se explica en 4.1.2. A continuación, se muestran gráficos de resultados para los distintos desfases de entrenamiento, en los que se grafican los resultados del pronóstico, versus, lo datos reales filtrados. Todos los valores graficados se encuentran des-normalizados.

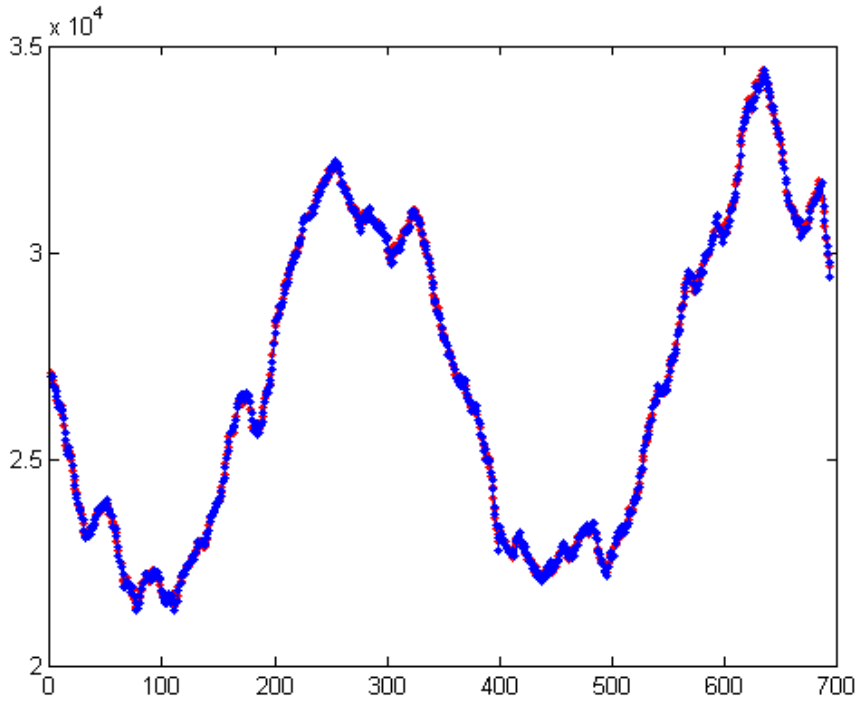


Figura 5.12 Predicción de datos diarios filtrados versus datos reales filtrados con desfase de entrenamiento 0

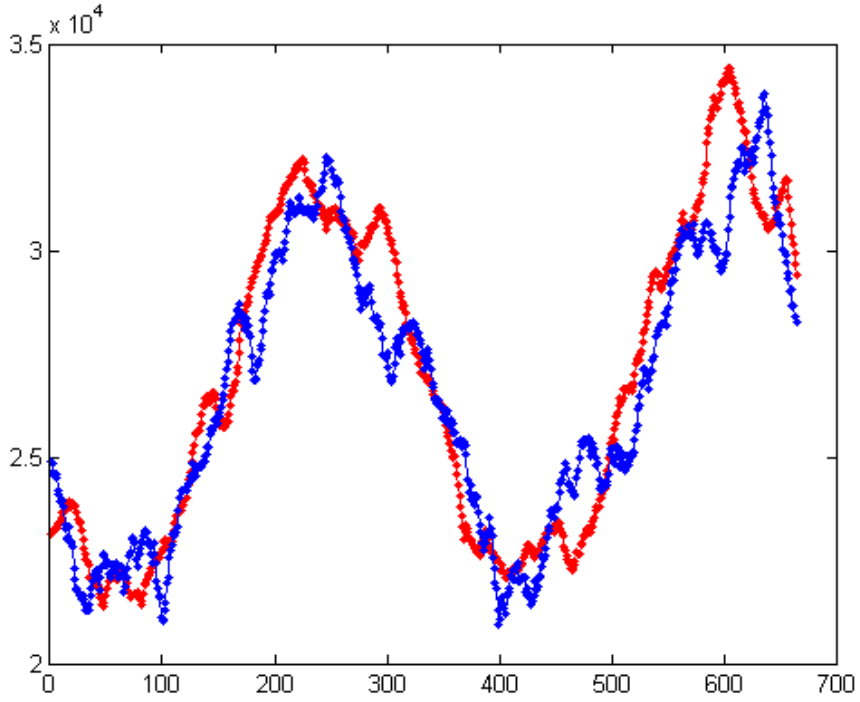


Figura 5.13 Predicción de datos diarios filtrados versus datos reales filtrados con desfase de entrenamiento 30 días

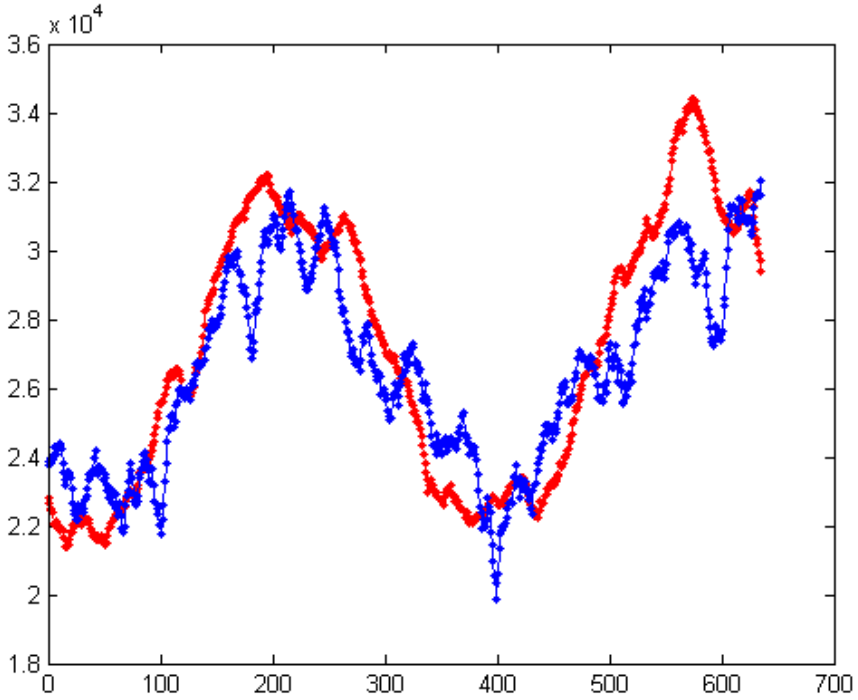


Figura 5.14 Predicción de datos diarios filtrados versus datos reales filtrados con desfase de entrenamiento 60 días

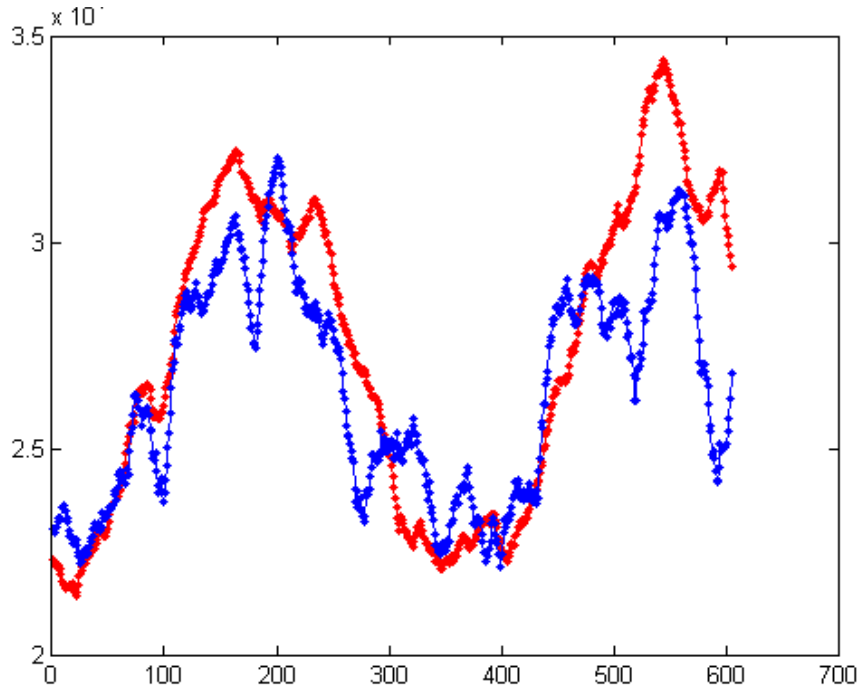


Figura 5.15 Predicción de datos diarios filtrados versus datos reales filtrados con desfase de entrenamiento 90 días

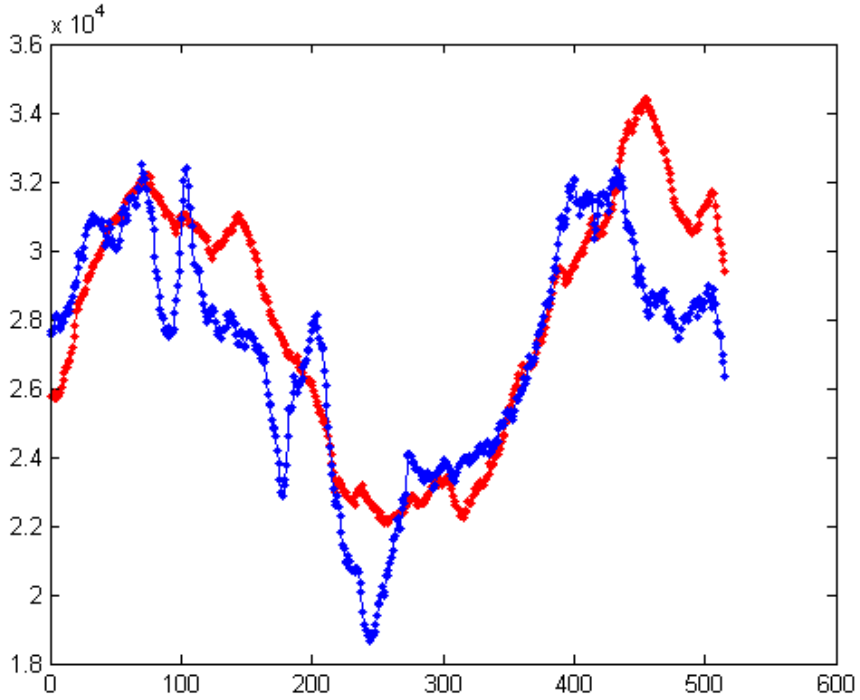


Figura 5.16 Predicción de datos diarios filtrados versus datos reales filtrados con desfase de entrenamiento 180 días

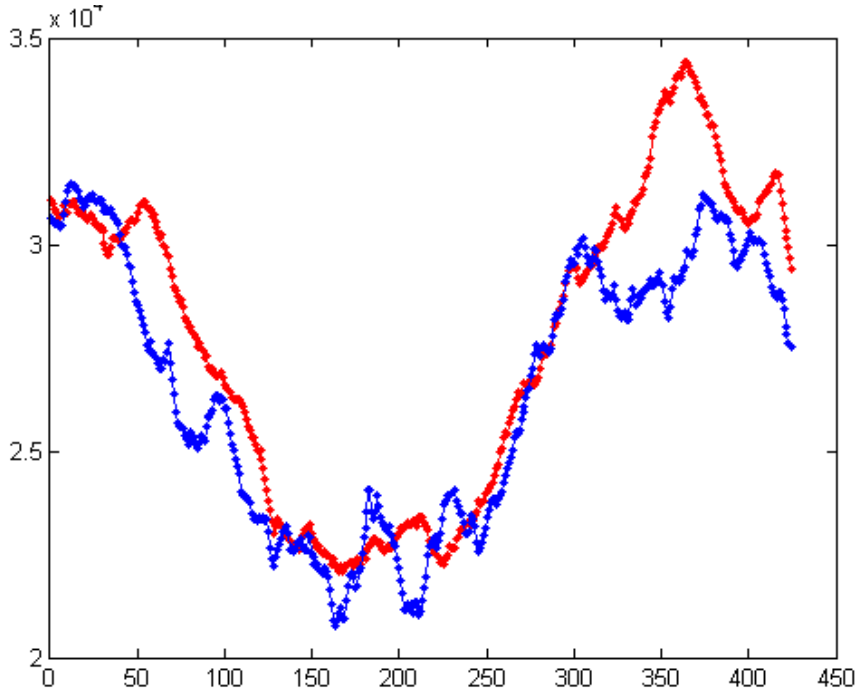


Figura 5.17 Predicción de datos diarios filtrados versus datos reales filtrados con desfase de entrenamiento 270 días

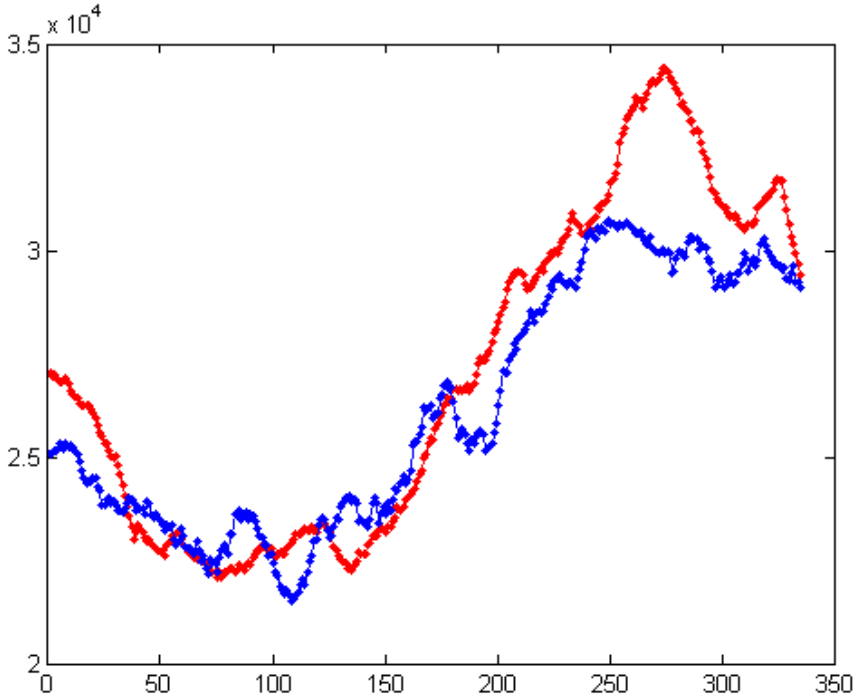


Figura 5.18 Predicción de datos diarios filtrados versus datos reales filtrados con desfase de entrenamiento 360 días

Las figuras nos muestran, en rojo, los datos reales filtrados, y en azul, el pronóstico de la red entrenada con datos filtrados. Como se puede observar, al igual que en el caso anterior, los datos pronosticados nunca pierden la forma cíclica ni la amplitud de los datos reales.

En la Tabla 5.8 se presentan los parámetros de evaluación de cada una de las redes diseñadas, correspondientes a los gráficos expuestos anteriormente. Adicionalmente, se muestra la arquitectura de cada una de las redes, especificando la cantidad de neuronas en la capa de entrada, oculta y de salida.

DESFASE (DÍAS)	MSE	MAPE (%)	MSPE (%)	ARQUITECTURA
0	$1.3536*10^{-5}$	0.2688	0.0012	56/30/1
30	0.0031	3.9880	0.2441	56/30/1
60	0.0057	5.4917	0.4450	56/30/1

90	0.0075	5.7608	0.5301	56/30/1
180	0.0075	5.9801	0.5492	56/30/1
270	0.0053	4.7212	0.3622	56/30/1
360	0.0041	4.4269	0.2815	56/30/1

Tabla 5.8 Errores y arquitectura de cada red entrenada con datos diarios filtrados

Los resultados para este caso, en que las redes son entrenadas con datos filtrados, son notablemente mejores que en el caso contrario, ya que, al filtrar los datos, estos pierden dispersión, y por lo tanto, es más sencillo para la red aprenda el comportamiento de consumo de agua.

A medida que los desfases de entrenamiento se van incrementando, los errores de predicción van aumentando hasta llegar a los 180 días de desfase. El desempeño de la red debiera comenzar a mejorar a partir del desfase de entrenamiento de 180 días ya que representa una proporción simétrica del periodo de los datos reales, tal como ocurre con la red de 360 días de desfase, que representan un periodo. Se considera que esto es una excepción y se atribuye a la aleatoriedad de la inicialización de la red al entrenarla.

Hay que tener en cuenta, que estos errores son calculados respecto a los datos originales filtrados, por lo tanto, un análisis, de que tan bien esta red es capaz de generar una predicción respecto a los datos reales sin filtrar, se verá en una próxima sección.

Respecto a la arquitectura de estas redes, al igual que en el caso del entrenamiento con datos no filtrados, se mantiene la misma cantidad de neuronas por capa para todos los casos. El tamaño de la red es escogido a través del entrenamiento de redes del mismo tipo, con diferentes arquitecturas, arrojando que la red de arquitectura *56/30/1* tiene el mejor desempeño.

5.1.4. Resultados mensuales

Al igual que la sección anterior, se presentan gráficos de la simulación de cada una de las redes neuronales diseñadas para cada desfase de entrenamiento de datos mensuales de demanda de agua y temperaturas máximas. Los datos mostrados en las gráficas se encuentran en unidades de litros/mes, es decir, la información ya ha sido des-normalizada.

Adicionalmente, se debe tener en cuenta que el entrenamiento de este tipo de redes se ha hecho con el 70% de los datos (al contrario del caso diario en el que se utiliza el 80% de los datos para su entrenamiento). Se decide tomar esta medida, para que el conjunto de datos de prueba sea suficiente para simular un año de predicciones, sin ocupar datos utilizados en el entrenamiento de la red.

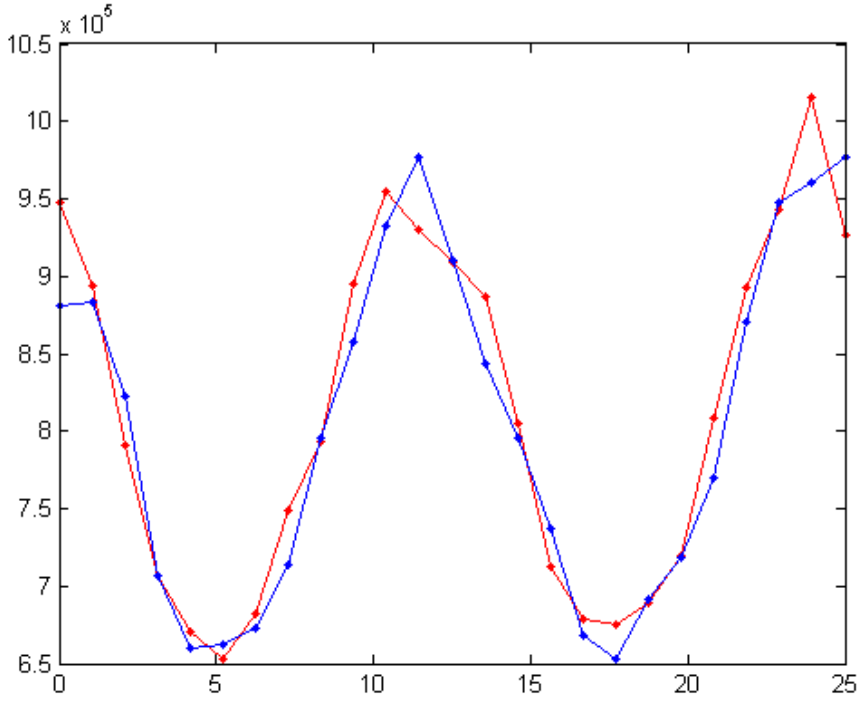


Figura 5.19 Predicción de datos mensuales de demanda de agua versus datos reales con desfase de entrenamiento 0

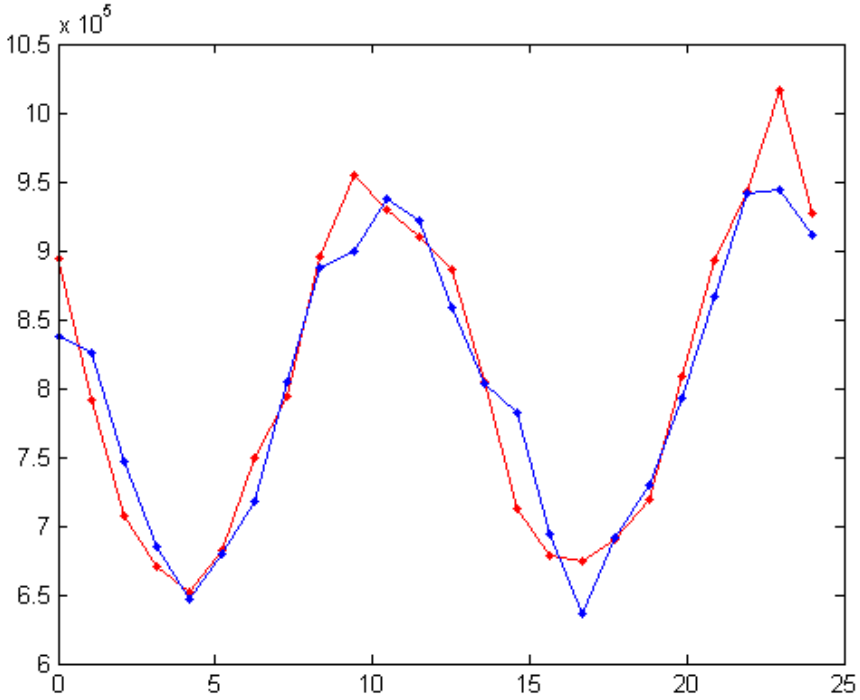


Figura 5.20 Predicción de datos mensuales de demanda de agua versus datos reales con desfase de entrenamiento de 1 mes

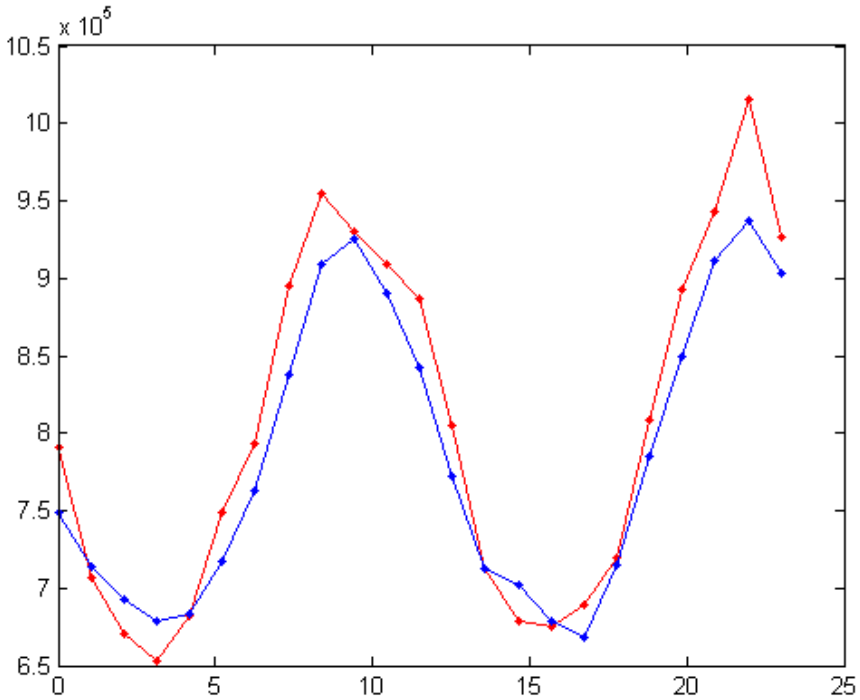


Figura 5.21 Predicción de datos mensuales de demanda de agua versus datos reales con desfase de entrenamiento de 2 meses

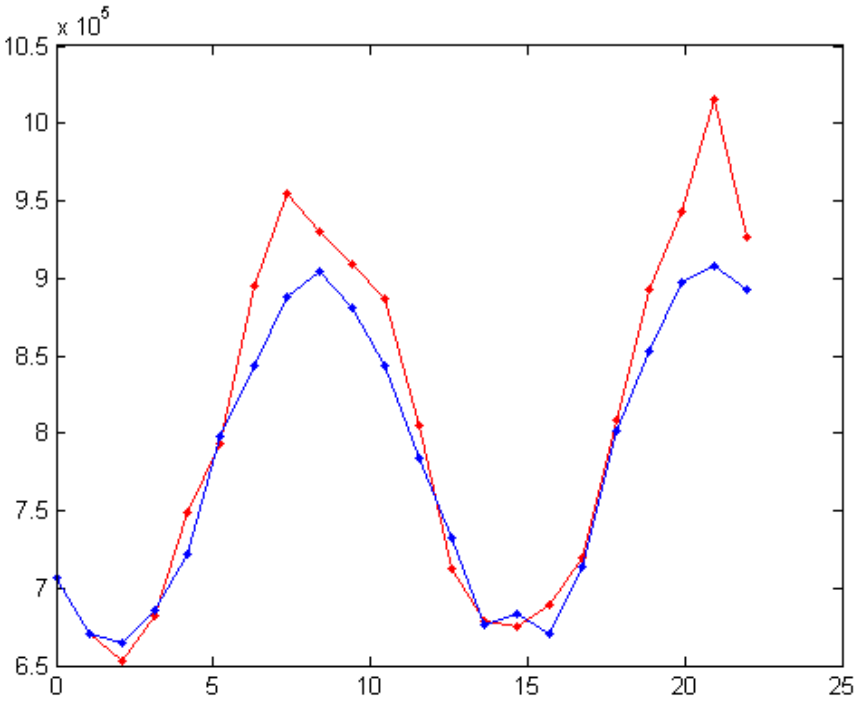


Figura 5.22 Predicción de datos mensuales de demanda de agua versus datos reales con desfase de entrenamiento de 3 meses

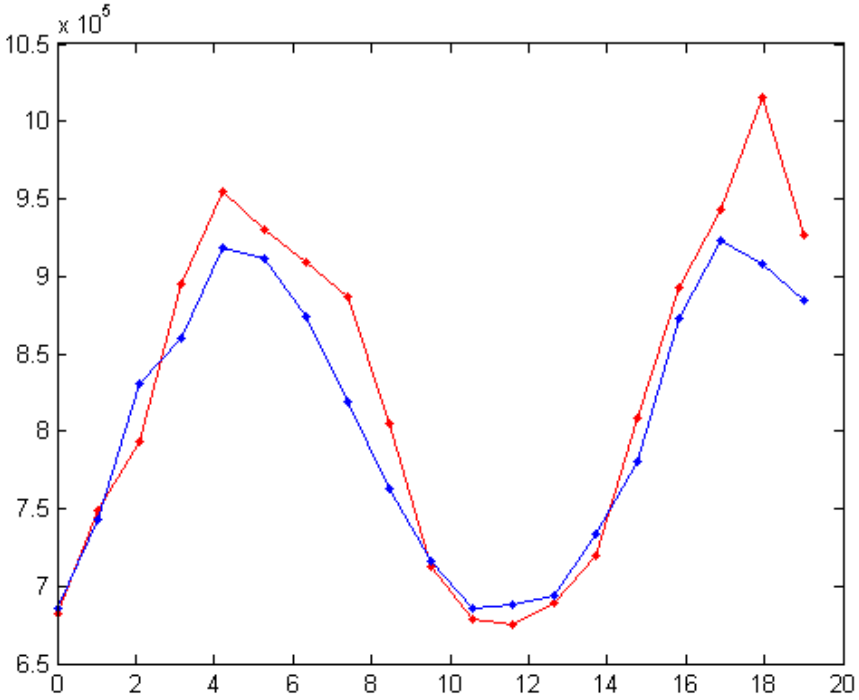


Figura 5.23 Predicción de datos mensuales de demanda de agua versus datos reales con desfase de entrenamiento de 6 meses

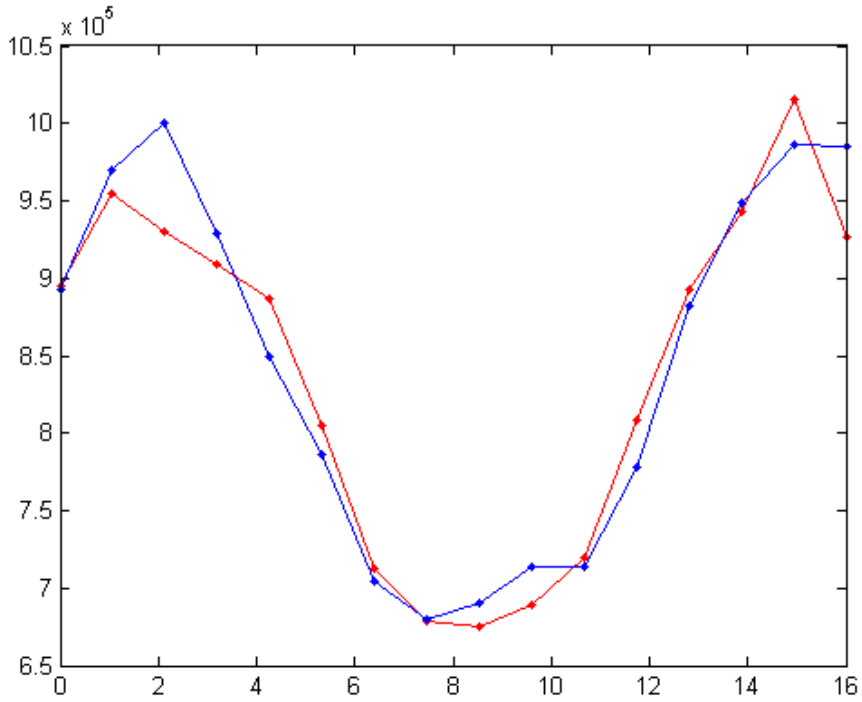


Figura 5.24 Predicción de datos mensuales de demanda de agua versus datos reales con desfase de entrenamiento de 9 meses

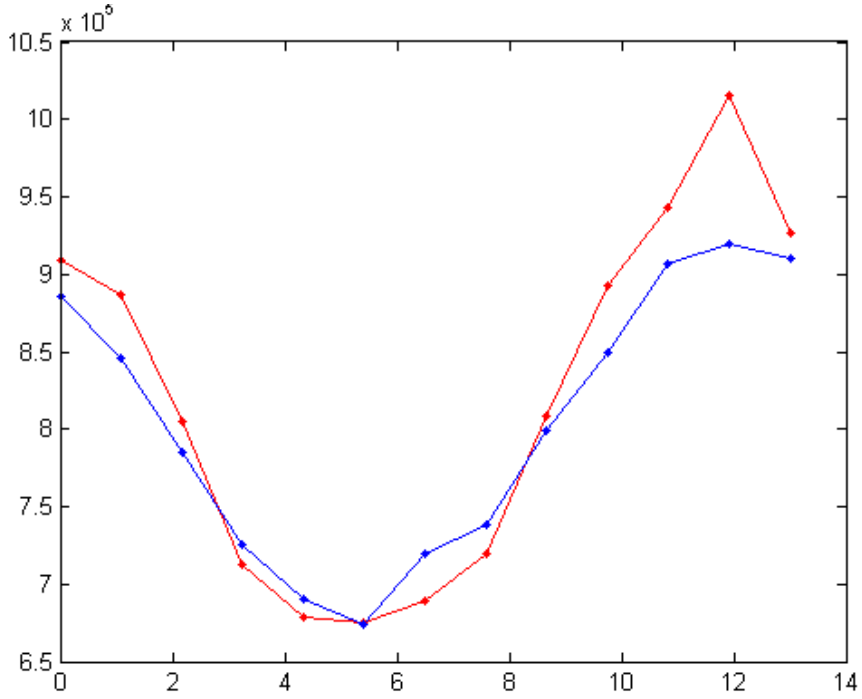


Figura 5.25 Predicción de datos mensuales de demanda de agua versus datos reales con desfase de entrenamiento de 12 meses

En las figuras se observa, en rojo, los datos mensuales reales de demanda de agua, y en azul, los datos mensuales pronosticados. Al igual que el caso de datos diarios, los datos pronosticados son capaces de seguir el comportamiento estacional y amplitud de los datos reales.

DESFASE (MESES)	MSE	MAPE (%)	MSPE (%)	ARQUITECTURA
0	0.0016	2.6534	0.1122	14/9/1
1	0.0018	2.9580	0.1521	14/9/1
2	0.0019	3.2165	0.1442	14/9/1
3	0.0024	2.9513	0.1519	14/9/1
6	0.0026	3.2396	0.1691	14/9/1
9	0.0015	2.5666	0.1071	14/9/1
12	0.0023	3.1768	0.1529	14/9/1

Tabla 5.9 Errores y arquitectura de cada red entrenada con datos mensuales sin filtrar

La Tabla 5.9, nos muestra los parámetros de evaluación y arquitecturas de todas las redes entrenadas con datos mensuales. En este caso, los errores de las redes no presentan una tendencia tan clara de aumento con el incremento del desfase de entrenamiento. Esto puede deberse a la notable disminución de datos, respecto al caso diario, en las que el desfase de entrenamiento era mucho mayor. Por lo tanto, se estima, que a pesar de que se entrena a la red con menos datos, la red, igualmente, es capaz de aprender el comportamiento de estos, por lo tanto, pedirle un desfase de 1 hasta 12 datos, comparado con un desfase de 30 hasta 360 datos, es una meta mucho más cercana para la red, y es por esta razón, que los errores de esta tabla, no presentan un incremento al aumentar los meses de desfase.

Otra causa para la inexistencia de una tendencia clara de aumento o disminución del error de predicción, puede deberse a que, al transformar 3742 datos diarios a 124 datos mensuales, la red no sea capaz de percibir tan claramente el comportamiento cíclico de los datos mensuales como en el caso diario.

Respecto a la arquitectura de las redes mencionadas, al igual que en los casos anteriores, se mantiene la misma cantidad de neuronas por capa. El tamaño de la red es escogido a través del entrenamiento de redes del mismo tipo, con diferentes arquitecturas, arrojando que la red de arquitectura *14/9/1* tiene el mejor desempeño.

5.2. Resultados comparativos

En esta sección, se presenta un experimento de predicción, en el que se pronostica un periodo anual definido entre abril del 2014 a marzo del 2015. Se debe tener en consideración que los datos de entrenamiento de todas las redes diseñadas no corresponden al periodo a pronosticar, ni tampoco a los datos necesarios para simular la predicción.

El objetivo de este experimento de predicción es evaluar el desempeño de las redes neuronales diseñadas, para poder comparar los 3 tipos de redes, (redes que utilizan datos diarios, mensuales y diarios filtrados), en un mismo experimento y obtener errores de un mismo set de datos.

A continuación, se procede a generar la simulación de cada una de las redes neuronales disponibles y se entregan los resultados correspondientes.

En la siguiente tabla los resultados son presentados en litros correspondientes a cada mes del periodo mencionado.

MES	DEMANDA REAL	DEMANDA PRONOSTICADA	ERROR (%)	DESFASE DE RED UTILIZADA
Abril	886393	895344	1.0099	0
Mayo	804720	807221	0.3109	360
Junio	712970	728830	2.2245	90
Julio	678374	691982	2.0059	270
Agosto	674805	700460	3.8018	270

septiembre	689542	715717	3.7960	270
octubre	719492	717270	0.3088	270
noviembre	808003	807713	0.0359	270
diciembre	892626	882664	1.1160	360
enero	942628	943429	0.0850	360
febrero	1015958	922032	9.2451	360
marzo	927059	887831	4.2314	360

Tabla 5.10 Tabla de resultados de red entrenada con datos diarios

MES	DEMANDA REAL	DEMANDA PRONOSTICADA	ERROR (%)	DESFASE DE RED UTILIZADA
abril	855563	856100	0.0628	0
mayo	782981	792026	1.1552	60
junio	694880	706000	1.6003	360
julio	671023	680459	1.4062	270
agosto	688458	705029	2.4069	360
septiembre	685546	720173	5.0510	270
octubre	748141	756439	1.1092	270
noviembre	837591	821647	1.9036	270
diciembre	900489	923867	2.5962	360
enero	979674	936948	4.3612	360
febrero	985812	843967	14.3887	360
marzo	929199	835894	10.0415	360

Tabla 5.11 Tabla de resultados de red entrenada con datos diarios filtrados

MES	DEMANDA REAL	DEMANDA PRONOSTICADA	ERROR (%)	DESFASE DE RED UTILIZADA
Abril	886393	857073	3.3078	1
Mayo	804720	809302	0.5694	1
Junio	712970	713913	0.1323	2
Julio	678374	678033	0.0504	9
Agosto	674805	684380	1.4189	6
Septiembre	689542	683952	0.8107	6
Octubre	719492	719772	0.0389	9
Noviembre	808003	803323	0.5792	12
Diciembre	892626	873155	2.1814	9
Enero	942628	928269	1.5232	9
Febrero	1015958	915147	9.9228	12
Marzo	927059	914015	1.4070	12

Tabla 5.12 Tabla de resultados de red entrenada con datos mensuales

Cada uno de los meses del periodo seleccionado son pronosticados con todas las opciones de redes disponibles de distintos desfases de entrenamiento, para luego proceder a seleccionar la red del desfase que mejor resultado entrega.

Adicionalmente, las redes elegidas para cada uno de los pronósticos se basan en que los datos de demanda entre abril del 2014 y marzo del 2015 son desconocidos, por lo tanto, no se puede utilizar cualquiera de las redes de distintos desfases para todos los meses pronosticados. Por ejemplo, para pronosticar marzo del 2015, solo se puede utilizar una red con desfase de 12 meses, ya que, si se utiliza una red con menor desfase, se necesitaría un dato correspondiente al periodo que se quiere pronosticar.

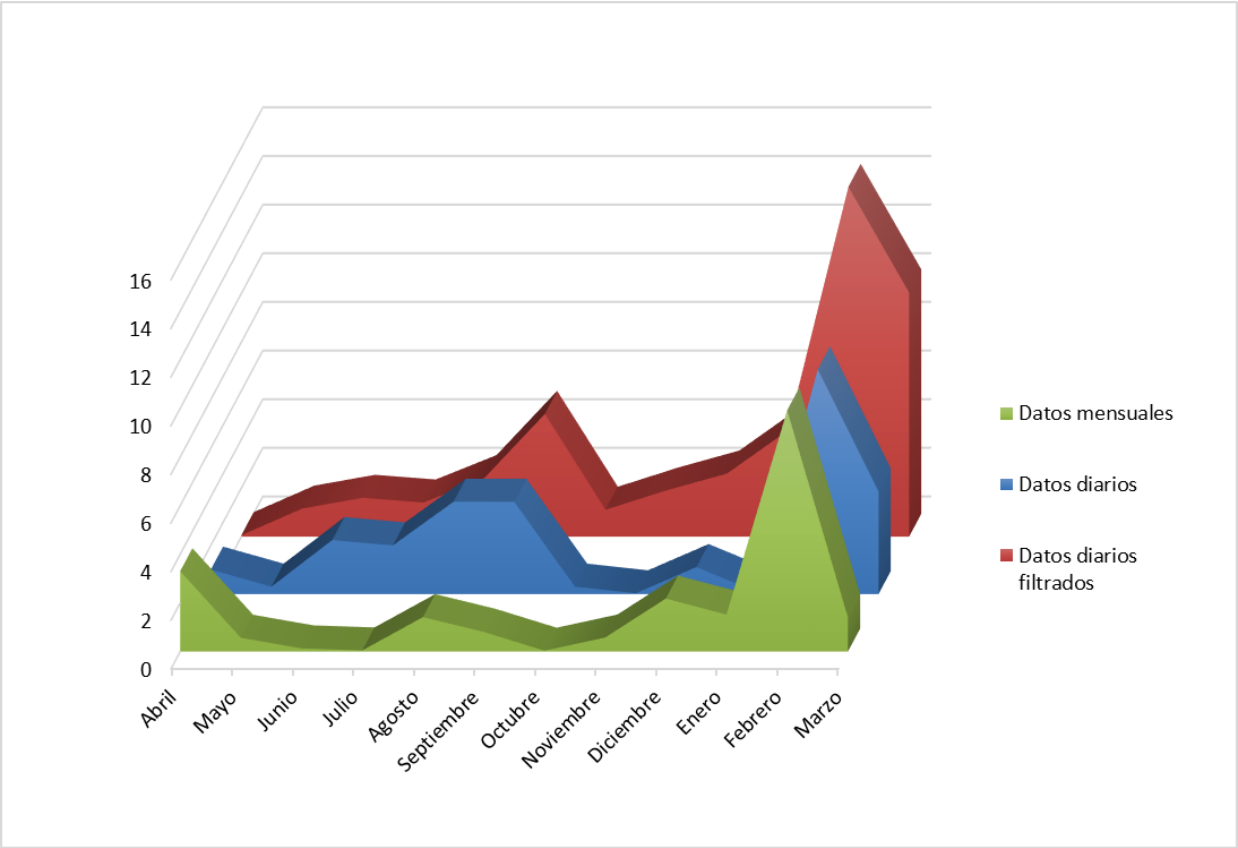


Figura 5.26 Errores de predicción para periodo de prueba entre abril 2014 y marzo 2015

La Figura 5.26 muestra los errores de predicción de cada mes pronosticado, los cuales se encuentran representados como errores porcentuales absolutos.

Se observa que las redes neuronales entrenadas con datos mensuales tienen, de forma generalizada, mejor desempeño de predicción en comparación a las redes entrenadas con datos diarios y datos diarios filtrados.

DATOS	ERROR PROMEDIO (%)	DESVIACION ESTANDAR	MÍNIMO ERROR (%)	MÁXIMO ERROR (%)
DIARIOS	2.3476	8.8059	0.0359	9.2451
DIARIOS FILTRADOS	3.8402	14.0959	0.0628	14.3887
MENSUALES	1.8285	9.0384	0.0389	9.9228

Tabla 5.13 Análisis de errores de predicción

Se aprecia en la tabla que no existe mucha diferencia entre el comportamiento del error de las redes entrenadas con datos diarios y datos mensuales. Si observamos los datos del gráfico de errores de predicción y los criterios de la tabla anterior, se tiene que la mejor opción es el entrenamiento de redes con datos mensuales. Ambas opciones poseen una dispersión similar, pero el valor medio de los errores nos indica que el caso mensual es el más indicado.

Definitivamente, descartar la opción de entrenar las redes neuronales con datos diarios filtrados es acertado, ya que todos los criterios de evaluación presentados en la Tabla 5.13 son los menos satisfactorios para el uso de estos datos. Adicionalmente, el gráfico de la Figura 5.26 muestra, igualmente, que es la opción más desfavorable.

Conclusiones

El objetivo específico de este trabajo es realizar un pronóstico de la demanda de producción de agua en las localidades de Los Andes, Calle Larga y Real Curimón de la provincia de los Andes. La utilidad de la predicción es lograr contabilizar las pérdidas de agua (ANC) mediante la diferencia entre dicha predicción y el pronóstico de la facturación que la empresa sanitaria dispone. Al no disponer de la predicción de pérdidas, la empresa queda sin herramientas para tomar una decisión respecto a la pérdida de agua, persistiendo y aumentando las consecuencias de no trabajar el agua no contabilizada, lo cual se traduce en una falla del servicio de abastecimiento, una pérdida en la inversión de producción, y finalmente, un eventual aumento en la contribución monetaria del cliente.

Respecto a la herramienta diseñada, se concluye que las redes diseñadas en base a datos de frecuencia mensual arrojan los mejores resultados. Por lo tanto, este trabajo propone el uso de una herramienta de predicción compuesta de una serie de redes neuronales entrenadas con distintos desfases de entrenamiento. Se considera que, para realizar predicciones anuales, su mejor uso es pronosticar, mes a mes, simulando la red que entregue mejores resultados, para obtener así la mejor predicción que la herramienta puede entregar (ver simulaciones en sección 5.2).

Teniendo en cuenta la propiedad multivariable de las redes neuronales y considerando la disminución del error de predicción al incorporar la temperatura como una nueva variable, se concluye que la inclusión de una, o más, variables de entrenamiento puede mejorar el desempeño de la red neuronal. Es así, como la disponibilidad de nuevas variables, que interfieran en el comportamiento del consumo de agua, puede disminuir el error en la predicción de la demanda de agua, entregando a la red más antecedentes que la ayuden a entender el comportamiento, o patrón, de los datos.

Es importante tener en cuenta que a medida que pasa el tiempo, la empresa sanitaria logra disponer de un mayor historial de datos de agua producida, por lo tanto,

al volver a diseñar y entrenar la red con un número mayor de datos, se estima que las nuevas predicciones pueden alcanzar mejores resultados.

Recordando que las pérdidas de agua corresponden a la diferencia entre el agua producida por la planta y el agua facturada, se puede recomendar a la empresa un cambio en la forma de contabilizar el agua facturada. Como se explica en 1.3, el agua facturada mensual es medida manualmente en un periodo variable, lo cual impide tener una facturación total de un periodo específico. La medición digital de la facturación de agua, puede eliminar este problema, y permitir a la empresa sanitaria obtener mediciones certeras de agua facturada. Esto se traduce en predicciones más exactas de agua facturada, y, por lo tanto, predicciones más acertadas del volumen de agua no contabilizada.

Las redes neuronales desarrolladas realizan predicciones sin considerar eventos no proyectables, por lo tanto, es necesario que la población en cuestión tenga un comportamiento estable. En otros países, la predicción de demanda de agua y la contabilización de pérdidas de agua son muy relevantes, es por esto que existen planes reguladores de crecimiento de la población para poder controlar la estabilidad del comportamiento de la misma. A modo de acotación, el desarrollo de un plan regulador en nuestro país podría evitar una sobre exigencia de producción de agua a plantas diseñadas para abastecer niveles más bajos, y al mismo tiempo, permitir que los pronósticos de demanda sean válidos.

La empresa sanitaria al conocer los niveles de pérdidas de agua analiza la decisión de trabajar, o no, en la disminución de estas. Por lo tanto, considerando lo anterior, se plantea como recomendación la realización de un programa de reducción y control de agua no contabilizada, en las que se incorporen una serie de medidas, tales como, crear sistemas de medición en tuberías para identificación de fugas, generar planes de mantenimiento de la red de distribución, crear metodologías de análisis de fraude y conexiones ilegales, entre otras, generando así una disminución del volumen de agua no contabilizada.

Referencias

[1] L. Dailey Paulson, «RWL Water» RWL Water LLC, 19 Febrero 2016. [En línea]. <https://www.rwlwater.com/que-es-el-agua-no-contabilizada/?lang=es>. [Último acceso: 16 Marzo 2016].

[2] G. Ahumada Theoduloz, «Continuidad, calidad y agua no contabilizada en distribución» *Revista AIDIS,* Octubre, 2013.

[3] H. Demuth, M. Beale y M. Hagan, Neural Network Toolbox, 2009.

[4] C. Lee Ho y K. Cheol Park, Prediction of montly transition of the composition stock price index using recurrent back-propagation, Elsevier Science Publishers, 1992.

[5] C. Ulbritcht, «Multi-recurrent Networks for Traffic Forecasting» Austrian Research Institute for Artificial Intelligence, Vienna, Austria.

[6] C. Marzban y J. G. Stumpf, A Neural Network for Tornado Prediction Bassed on Doppler Radar-derived Attributes, National Severe Storms Laboratory's, 1998.

[7] G. Ríos, «Series de Tiempo» Departamento de Ciencias de la Computación. Universidad de Chile, Santiago, Chile, 2008.

[8] «MathWorks» The MathWorks, Inc., 2016. [En línea]. <http://es.mathworks.com/help/nnet/ug/improve-neural-network-generalization-and-avoid-overfitting.html>. [Último acceso: 21 Marzo 2016].

Anexo A

Método de gradiente descendente

A continuación, se procede a deducir las expresiones que definen el método del gradiente descendente aplicado a los algoritmos de backpropagation.

El objetivo general del método de gradiente descendente, es encontrar el valor de la variable independiente que minimice una función determinada, es decir, hallar el valor de x_0 que minimiza la función $F(x)$.

El primer paso, es hacer una puesta inicial de manera aleatoria, es decir inicializamos el x_0 en cualquier parte del espacio x. Luego se realiza un ciclo de iteraciones donde se trata de mejorar la puesta inicial, para posteriormente converger al valor que minimiza la función. Lo anterior obedece a la siguiente expresión.

$$x_0 = x_0 - \alpha \frac{dF}{dx} \quad (A.1)$$

$$x = x_0$$

Se debe notar que esta expresión necesita el cálculo del gradiente de la función F, con respecto a x, y evaluado en x_0, para poder actualizar su valor. Por otro lado, se tiene la constante α, que está multiplicando este gradiente para poder determinar, iteración tras iteración, cuánto va a moverse x_0 respecto a su valor anterior. El signo – en la expresión A.1 genera que en cada actualización iterativa de x_0 el gradiente de la función F, evaluado en x_0 y multiplicado por α, disminuya el valor anterior de x_0, para así poder llegar al menor valor de la función. Es allí donde se justifica el nombre del método de gradiente descendente.

Para tener claro cómo afecta la constante α, o también llamada tasa de aprendizaje, se puede observar en la Figura 8.1 cómo funciona el método según el valor de la tasa. Si se sigue el método en el caso en que α es pequeño, se puede observar que el método converge asintóticamente hasta llegar al mínimo de la función. En el caso en que α es mediano, se ve que el método también converge asintóticamente, pero con oscilaciones. Por último, cuando α es grande el método diverge, y, por lo tanto, no es útil para encontrar el mínimo de la función.

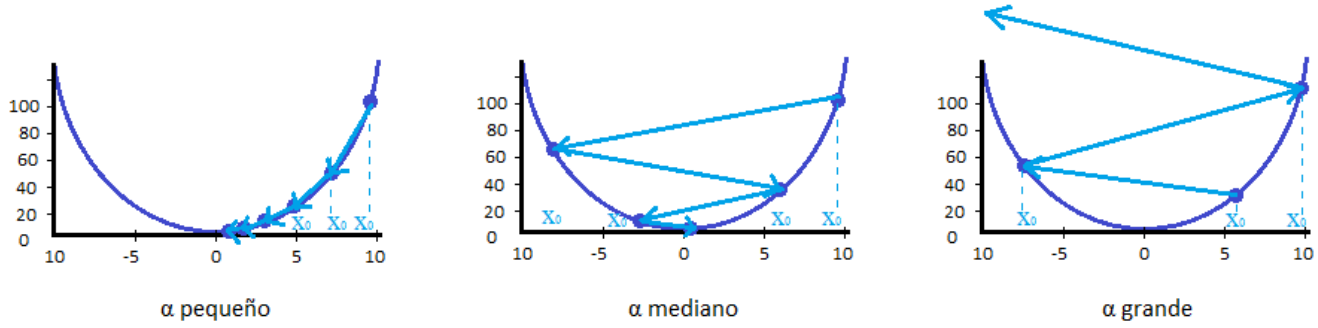


Figura 8.1 Efectos de la tasa de aprendizaje

En términos de las redes neuronales, el método del gradiente descendente, nos ayuda a encontrar el mínimo valor de la función del error entre la salida de la red y la salida real del sistema. Junto a esto, el algoritmo backpropagation puede actualizar los valores de los pesos sinápticos en el entrenamiento de la red.

Para explicar más a fondo el algoritmo, es necesario definir algunos parámetros como se muestra en la Figura 8.2.

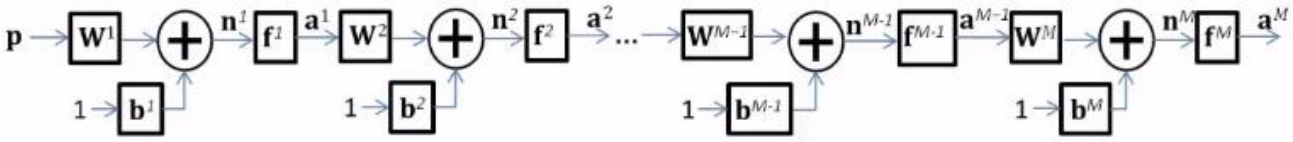


Figura 8.2 Parámetros de una red neuronal

Donde el conjunto de datos de entradas y salidas de la red son: $(p_1, t_1), (p_2, t_2), \dots, (p_q, t_q), \dots, (p_Q, t_Q)$.

La salida del sistema, luego del proceso de los datos de entrada por las M capas de la red es descrita por la ecuación recursiva A.2.

$$a^m = f^m(W^m\, a^{m-1} + b^m) \qquad m \in [1, \dots, M] \qquad (A.2)$$

$$a^0 = p \qquad (A.3)$$

Las matrices de pesos sinápticos W^m y valores de sesgo b^m, son representados por la ecuación 3.4.

$$X^m = [W^m\, b^m]^T \qquad (A.4)$$

El error de la red equivale a la diferencia entre la matriz de salida y la matriz de datos de salida del conjunto de datos, como se muestra a continuación en la ecuación A.5.

$$e_q = t_q - a_q^M \qquad (A.5)$$

La ecuación A.6 describe la función a minimizar, la cual está definida por el error cuadrático de la red, en función de los pesos sinápticos y valores de sesgo.

$$E(X^1, \dots, X^m) = e_q^T\, e_q \qquad (A.6)$$

En la expresión A.7 se ve la aplicación del método de gradiente descendente a la función de error cuadrático definida. Hay que tener en cuenta que, para utilizar este método como algoritmo de aprendizaje de la red, es preciso que la ecuación A.6, es decir, el error cuadrático, sea diferenciable.

$$X^m = X^m - \alpha \frac{dE}{dX^m} \qquad (A.7)$$

$$X^m = X_0^m$$

Para comenzar a desglosar la expresión A.7, se procede a trasponerla (expresión A.8), de tal forma, que luego de llegar a las expresiones finales de cada uno de sus elementos en términos conocidos, se puedan definir los pesos sinápticos W^m y valores de sesgo b^m por separado.

$$(X^m)^T = (X^m)^T - \alpha \frac{dE}{dX^m}^T \qquad (A.8)$$

La ecuación A.9 nos da la expresión para el gradiente del error cuadrático, que no es más que una matriz jacobiana en donde cada una de sus columnas corresponde a la variación del error cuadrático de las s^m neuronas de la capa m en función de X.

$$\frac{dE}{dX^m} = \left[\frac{dE}{dx_1^m} \ \ldots \ \frac{dE}{dx_{s^m}^m}\right] \qquad (A.9)$$

Para calcular la derivada del error cuadrático es necesario tener la expresión de cada uno de sus elementos. En la ecuación A.11 se muestra, mediante la regla de la cadena, una expresión para el gradiente del error de cada neurona de una cierta capa. Para ello hay que tener en cuenta la expresión A.10 que define la entrada neta de una neurona, que corresponde al valor numérico que es ingresado a la función de activación de cada neurona, tal como lo muestra la Figura 8.2. Por lo tanto, la entrada neta equivale al producto de los pesos y valores de sesgo con el vector de entradas aumentado.

$$n_i^m = x_i^T \, z^m \qquad (A.10)$$

$$\frac{dE}{dx_i^m} = \frac{dn_i^m}{dx_i^m} \frac{dE}{dn_i^m} = z^m s_i^m \qquad (A.11)$$

A modo de aclaración, el vector de entradas es aumentado en 1, como se ve en la ecuación A.12, para que el producto entre las entradas y los elementos de x^m incluyan la multiplicación entre las entradas y valores de sesgo de la neurona

$$z^m = \begin{pmatrix} a^{m-1} \\ 1 \end{pmatrix} \qquad (A.12)$$

Como se ve en la ecuación A.11, se observa la sigla s_i^m, la cual corresponde a la sensibilidad de la neurona, y por lo tanto representa la derivada del error cuadrático en función de la entrada neta de la neurona i de la capa m.

Por ahora, la expresión del gradiente del error cuadrático queda según la expresión A.13, donde se sustituyen los gradientes recién descritos en la matriz jacobiana. Tener en cuenta que s^m(sin subíndice) representa al vector de sensibilidades $[s_1^m \dots s_{s^m}^m]^T$ de cada una de las neuronas de la capa m.

$$\frac{dE}{dX^m} = [z^m s_1^m \dots z^m s_{s^m}^m] = z^m (s^m)^T = \begin{pmatrix} a^{m-1} \\ 1 \end{pmatrix} (s^m)^T = \begin{pmatrix} a^{m-1}(s^m)^T \\ (s^m)^T \end{pmatrix} \quad (A.13)$$

A continuación, en la ecuación A.14, reemplazamos el resultado de la expresión anterior en la ecuación del gradiente descendente traspuesto (ecuación A.8). Para luego expandirla con la definición de X^m, y así obtener las expresiones que representen a cada uno de sus elementos. La ecuación A.14 representa en realidad dos ecuaciones, por lo tanto, solo basta desacoplarlas para obtener los pesos sinápticos y valores de sesgo para cada capa (ecuación A.15 y A.16).

$$\frac{dE}{dX^m}^T = [s^m (a^{m-1})^T \quad s^m] \quad (A.13)$$

$$[W^m \; b^m] = [W^m \; b^m] - \alpha [s^m (a^{m-1})^T \quad s^m] \quad (A.14)$$

$$W^m = W^m - \alpha s^m (a^{m-1})^T \quad (A.15)$$

$$b^m = b^m - \alpha s^m \quad (A.16)$$

$$\forall m \in [1, \dots, M]$$

Para esclarecer aún más las ecuaciones obtenidas, es preciso tener una expresión para la sensibilidad de la neurona, en función de elementos conocidos. Esto a través de la obtención de una ecuación recursiva que permita calcular las sensibilidades de cada capa de la red neuronal.

Como se ve en la ecuación A.17, se ve una expresión que representa la derivada del error cuadrático en función de la entrada neta de la capa $m-1$. Se puede expresar esta derivada utilizando la regla de la cadena, en términos de la sensibilidad de la capa posterior. De esta manera, se tiene la variación del error en función de la entrada neta de la capa m, y la variación de la entrada neta de la capa m en función de la entrada posterior. Luego, utilizando la nomenclatura de la sensibilidad, se pueden sustituir las derivadas de la ecuación A.17 por eses para llegar a la ecuación recursiva A.18. Al igual que la sensibilidad, se debe tener en cuenta que n^m(sin subíndice) representa al vector de entradas netas $[n_1^m \dots n_{s^m}^m]^T$ de cada una de las neuronas de la capa m.

$$\frac{dE}{dn^m} = \frac{dn^m}{dn^{m-1}} \frac{dE}{dn^m} \qquad (A.17)$$

$$s^{m-1} = \frac{dn^m}{dn^{m-1}} \, s^m \qquad (A.18)$$

Se debe recordar que la expresión que define a las entradas netas de una capa, depende de la función de activación de la capa anterior y esta a su vez de sus entradas netas, como lo muestra la ecuación A.19. Con la ayuda de la regla de la cadena junto a la ecuación A.19 (expresión A.20) se puede llegar a una expresión para la derivada de la ecuación A.18. En ella se encuentra la derivada de la función de activación f^{m-1} en función de las entradas netas de la misma capa, la cual se representa con la letra $\dot{F}$ por ahora. Hasta el momento, la expresión para la sensibilidad queda según la ecuación A.21.

$$n^m = W^m f^{m-1}(n^{m-1}) + b^m \qquad (A.19)$$

$$\frac{dn^m}{dn^{m-1}} = \frac{df^{m-1}}{dn^{m-1}} \frac{dn^m}{df^{m-1}} = \dot{F}^{m-1}(n^{m-1}) \, (W^m)^T \qquad (A.20)$$

$$s^{m-1} = \dot{F}^{m-1}(n^{m-1}) \, (W^m)^T \, s^m \qquad (A.21)$$

$$\forall m \in [M, \dots, 2]$$

La última expresión es válida para todas las capas de la red, excepto para la capa de salida M. Para ello, se necesita conocer la sensibilidad de la última capa, y así poder comenzar el proceso de recursión, o retropropagación, de las sensibilidades de cada capa de la red. Por esta razón, se procede a calcular la sensibilidad de la última capa de la red.

Con la ayuda de la regla de la cadena, representamos la sensibilidad de la capa de salida mediante 3 derivadas, de las cuales, la última, puede ser resueltas recordando la ecuación A.6 que representa la definición del error cuadrático. Por otro lado, la segunda derivada, la variación del error de la red en función de la función de activación, se puede resolver mediante la ecuación A.22 que define error de la red mediante la resta entre el valor real y la salida de la red. Por último, la primera derivada, que representa la variación de la función de activación de la capa M en función de la salida neta de la misma capa, se simboliza con la letra $\dot{F}^m$, la cual, solo puede ser descrita de forma literal sabiendo exactamente cuál es la función de activación de la capa de salida. Hasta ahora, la expresión para la sensibilidad s^M queda según la ecuación A.24.

$$E = e_q^T\, e_q \qquad (A.6)$$

$$e_q = t_q - f^M(n^M) \qquad (A.22)$$

$$s^M = \frac{dE}{dn^M} = \frac{df^M}{dn^M}\,\frac{de_q}{df^M}\,\frac{dE}{de_q} = \frac{df^M}{dn^M}(-I)\big(2e_q\big) \qquad (A.23)$$

$$s^M = -2\,\dot{F}^m(n^m)\,e_q \qquad (A.24)$$

Desglosando un poco más la derivada de la función de activación con respecto a la entrada neta de la capa M, debemos tener en cuenta que la derivada nos entrega una matriz que contiene los gradientes de cada función de activación de la capa en función del vector de entradas netas, (recordar que n^m corresponde al vector de las entradas netas de todas las neuronas de la capa M). Por lo tanto, la derivada se define según la ecuación A.25 donde cada uno de sus elementos son descritos por la ecuación A.26.

$$\dot{F}^m(n^m) = \frac{df^m}{dn^m} = \left[\frac{df_1^m}{dn^m} \ \dots \ \frac{df_{s^m}^m}{dn^m}\right] \qquad (3.25)$$

$$\frac{df_i^m}{dn^m} = \frac{df_i^m}{dn_1^m} + \frac{df_i^m}{dn_2^m} + \cdots + \frac{df_i^m}{dn_i^m} + \cdots + \frac{df_i^m}{dn_{s^m}^m} \qquad (3.26)$$

Se debe tener en cuenta que las funciones de activación de cada neurona solo se ven afectadas por sus propias entradas netas, por lo tanto, $\dot{F}^m(n^m)$ queda como una matriz diagonal tal como se ve en la ecuación A.27. Hay que recordar que usualmente las funciones de activación suelen ser la misma para todas las neuronas de la misma capa, por lo tanto, la matriz de gradientes queda como la matriz diagonal de las derivadas de la función de activación respecto a cada entrada neta de la capa.

$$\frac{df^m}{dn^m} = \begin{bmatrix} \frac{df_1^m}{dn_1^m} & 0 & \cdots & 0 \\ 0 & \frac{df_2^m}{dn_2^m} & \cdots & 0 \\ & & & \vdots \\ \vdots & \vdots & \ddots & \\ 0 & 0 & \cdots & \frac{df_{s^m}^m}{dn_{s^m}^m} \end{bmatrix} = diag\left(\frac{df_i^m}{dn_i^m}\right) \qquad (3.27)$$

Finalmente, solo queda definir las derivadas de cada una de las posibles funciones de activación definidas en 0.

A continuación, la ecuación A.28 representa la función de activación logarítmica sigmoidal de la capa i, mientras que, la ecuación A.29, muestra su derivada en términos de la salida de la neurona. La ecuación A.30, representa la expresión resultante para la función $\dot{F}^m(n^m)$ considerando que todas las neuronas de la capa m tienen la función de activación logarítmica sigmoidal. Al igual que las ecuaciones planteadas para la función logarítmica sigmoidal, las ecuaciones A.31, A.32 y A.33 son referidas a la función de activación tangencial sigmoidal y las ecuaciones A.34, A.35 y A.36, lo son para la función de activación lineal.

$$a_i = f_i(n_i) = \frac{1}{1 + e^{-n_i}} \qquad (A.28)$$

$$\frac{df}{dn^i} = (1 - a_i)\, a_i \quad (A.29)$$

$$\dot{F}^m(n^m) = diag\left((1 - a_i^m)\, a_i^m\right) \quad (A.30)$$

$$a_i = f_i(n_i) = \frac{e^{n_i} - e^{-n_i}}{e^{n_i} + e^{-n_i}} \quad (A.31)$$

$$\frac{df}{dn^i} = 1 - a_i^2 \quad (A.32)$$

$$\dot{F}^m(n^m) = diag(1 - (a_i^m)^2) \quad (A.33)$$

$$a_i = f_i(n_i) = n_i \quad (A.34)$$

$$\frac{df}{dn^i} = 1 \quad (A.35)$$

$$\dot{F}^m(n^m) = diag(1) = I \quad (A.36)$$

A continuación, se muestra un resumen de lo anteriormente explicado para poder tener más claro el algoritmo backpropagation.

Recordar que el conjunto de ejemplos para el entrenamiento de la red es: $(\mathrm{p}_1, \mathrm{t}_1), (\mathrm{p}_2, \mathrm{t}_2), \dots, (\mathrm{p}_\mathrm{q}, \mathrm{t}_\mathrm{q}), \dots, (\mathrm{p}_\mathrm{Q}, \mathrm{t}_\mathrm{Q})$, con Q siendo la cantidad total de ejemplos. Teniendo en cuenta que por cada ejemplo p_q la red de M capas genera una salida a_q^M, y que las matrices de pesos y valores de sesgo se inicializan aleatoriamente, el algoritmo se resume a lo siguiente:

$for \;\; epocas = 1 : N_{epocas}$

{ $for \;\; q = 1 : Q$

{ Propagar hacia adelante

$a^m = f^m(W^m a^{m-1} + b^m) \qquad \forall m$

Propagar hacia atrás

$e_q = t_q - a_q^M$

$s^M = -2\dot{F}^M(n^M)\, e^q$

$s^{m-1} = \dot{F}^{m-1}(n^{m-1})(W^m)^T \qquad \forall m \in [M, \dots, 2]$

Actualizar pesos y valores de sesgo

$W^m = W^m - \alpha s^m (a^{m-1})^T$

$b^m = b^m - \alpha s^m$ }

end }

end

Anexo B

Método de gradiente conjugado

A continuación, se procede a entregar una descripción del método de gradiente conjugado para un mejor entendimiento de los algoritmos avanzados de backpropagation mencionados en la sección 2.1.

El objetivo general del método de gradiente conjugado es encontrar el valor de la variable independiente que minimice una función determinada.

Para explicar este método de optimización vamos a suponer que todos los pesos sinápticos óptimos de la red están representados por los elementos del vector x, el cual obedece al siguiente sistema de ecuaciones:

$$Ax = b \quad (B.1)$$

Donde A es una matriz simétrica y definida positiva.

Sea el producto interno definido por:

$$u^T A\, v = \langle u, v \rangle \quad (B.2)$$

Si $\langle u, v \rangle = 0$, se considera que los vectores u y v, no nulos, son conjugados entre si respecto a la matriz A.

Sea $\{p_k\}$ una secuencia de n vectores conjugados entre si respecto a la matriz A como lo indica la expresión B.3, donde todos los vectores ortogonales p_i forman una base de R^n.

$$\langle p_i, p_j \rangle = 0 \quad (B.3)$$

$$i \neq j$$

Entonces el vector x se puede escribir como una combinación lineal de los vectores p_i de acuerdo a la siguiente expresión:

$$x = \sum_{i=1}^{n} \alpha_i \, p_i = \alpha_k \, p_k \quad (B.4)$$

Luego, el valor del coeficiente α_k se obtiene multiplicando ambos lados de la expresión B.4 por la matriz A y el vector p_k^T.

$$b = Ax = \sum_{i=1}^{n} \alpha_i \, A \, p_i \quad (B.5)$$

$$p_k^T \, b = \alpha_k \, p_k^T \, A \, p_k \quad (B.6)$$

Donde finalmente obtenemos una expresión para α_k.

$$\alpha_k = \frac{p_k^T \, b}{p_k^T \, A \, p_k} = \frac{\langle p_k, b \rangle}{\|p_k\|_A^2} \quad (B.7)$$

Anexo C

Códigos en Matlab

A continuación, se presentan los códigos más relevantes para el desarrollo de las redes predictivas.

```
% EXTRACCIÓN DE DATOS DESDE ARCHIVO EXCEL

function [LosAndesDDA, LosAndesTmax, LosAndesTmin, Promedio] =
Datos_Esval(argFrec, temp)

if (temp == '1')  % 1
        % DEMANDA
        % El Sauce
        ElSauce2005 = xlsread('Recopilacion datos ESVAL.xlsx', 'El
Sauce', 'D7:D371');
        ElSauce2006 = xlsread('Recopilacion datos ESVAL.xlsx', 'El
Sauce', 'J7:J371');
        ElSauce2007 = xlsread('Recopilacion datos ESVAL.xlsx', 'El
Sauce', 'P7:P371');
        ElSauce2008 = xlsread('Recopilacion datos ESVAL.xlsx', 'El
Sauce', 'V7:V372');
        ElSauce2009 = xlsread('Recopilacion datos ESVAL.xlsx', 'El
Sauce', 'AB7:AB371');
        ElSauce2010 = xlsread('Recopilacion datos ESVAL.xlsx', 'El
Sauce', 'AH7:AH371');
        ElSauce2011 = xlsread('Recopilacion datos ESVAL.xlsx', 'El
Sauce', 'AN7:AN371');
        ElSauce2012 = xlsread('Recopilacion datos ESVAL.xlsx', 'El
Sauce', 'AT7:AT372');
        ElSauce2013 = xlsread('Recopilacion datos ESVAL.xlsx', 'El
Sauce', 'AZ7:AZ371');
        ElSauce2014 = xlsread('Recopilacion datos ESVAL.xlsx', 'El
Sauce', 'BF7:BF371');
        ElSauce2015 = xlsread('Recopilacion datos ESVAL.xlsx', 'El
Sauce', 'BL7:BL96');

        % Bellavista
        Bellavista2005 = xlsread('Recopilacion datos ESVAL.xlsx',
'Bellavista', 'D7:D371');
        Bellavista2006 = xlsread('Recopilacion datos ESVAL.xlsx',
'Bellavista', 'J7:J371');
        Bellavista2007 = xlsread('Recopilacion datos ESVAL.xlsx',
'Bellavista', 'P7:P371');
        Bellavista2008 = xlsread('Recopilacion datos ESVAL.xlsx',
'Bellavista', 'V7:V372');
        Bellavista2009 = xlsread('Recopilacion datos ESVAL.xlsx',
'Bellavista', 'AB7:AB371');
        Bellavista2010 = xlsread('Recopilacion datos ESVAL.xlsx',
'Bellavista', 'AH7:AH371');
```

```
        Bellavista2011 = xlsread('Recopilacion datos ESVAL.xlsx',
'Bellavista', 'AN7:AN371');
        Bellavista2012 = xlsread('Recopilacion datos ESVAL.xlsx',
'Bellavista', 'AT7:AT372');
        Bellavista2013 = xlsread('Recopilacion datos ESVAL.xlsx',
'Bellavista', 'AZ7:AZ371');
        Bellavista2014 = xlsread('Recopilacion datos ESVAL.xlsx',
'Bellavista', 'BF7:BF371');
        Bellavista2015 = xlsread('Recopilacion datos ESVAL.xlsx',
'Bellavista', 'BL7:BL96');

        % Miraflores
        Miraflores2005 = xlsread('Recopilacion datos ESVAL.xlsx',
'Miraflores', 'D7:D371');
        Miraflores2006 = xlsread('Recopilacion datos ESVAL.xlsx',
'Miraflores', 'J7:J371');
        Miraflores2007 = xlsread('Recopilacion datos ESVAL.xlsx',
'Miraflores', 'P7:P371');
        Miraflores2008 = xlsread('Recopilacion datos ESVAL.xlsx',
'Miraflores', 'V7:V372');
        Miraflores2009 = xlsread('Recopilacion datos ESVAL.xlsx',
'Miraflores', 'AB7:AB371');
        Miraflores2010 = xlsread('Recopilacion datos ESVAL.xlsx',
'Miraflores', 'AH7:AH371');
        Miraflores2011 = xlsread('Recopilacion datos ESVAL.xlsx',
'Miraflores', 'AN7:AN371');
        Miraflores2012 = xlsread('Recopilacion datos ESVAL.xlsx',
'Miraflores', 'AT7:AT372');
        Miraflores2013 = xlsread('Recopilacion datos ESVAL.xlsx',
'Miraflores', 'AZ7:AZ371');
        Miraflores2014 = xlsread('Recopilacion datos ESVAL.xlsx',
'Miraflores', 'BF7:BF371');
        Miraflores2015 = xlsread('Recopilacion datos ESVAL.xlsx',
'Miraflores', 'BL7:BL96');

        ElSauce = vertcat(ElSauce2005, ElSauce2006, ElSauce2007,
ElSauce2008, ElSauce2009, ElSauce2010, ElSauce2011, ElSauce2012,
ElSauce2013, ElSauce2014, ElSauce2015);
        Bellavista = vertcat(Bellavista2005, Bellavista2006,
Bellavista2007, Bellavista2008, Bellavista2009, Bellavista2010,
Bellavista2011, Bellavista2012, Bellavista2013, Bellavista2014,
Bellavista2015);
        Miraflores = vertcat(Miraflores2005, Miraflores2006,
Miraflores2007, Miraflores2008, Miraflores2009, Miraflores2010,
Miraflores2011, Miraflores2012, Miraflores2013, Miraflores2014,
Miraflores2015);

        % TEMPERATURA
        Tmax2005 = xlsread('Recopilacion datos ESVAL.xlsx',
'Temperatura', 'C7:C371');
        Tmax2006 = xlsread('Recopilacion datos ESVAL.xlsx',
'Temperatura', 'H7:H371');
        Tmax2007 = xlsread('Recopilacion datos ESVAL.xlsx',
'Temperatura', 'M7:M371');
```

```
        Tmax2008 = xlsread('Recopilacion datos ESVAL.xlsx',
'Temperatura', 'R7:R372');
        Tmax2009 = xlsread('Recopilacion datos ESVAL.xlsx',
'Temperatura', 'W7:W371');
        Tmax2010 = xlsread('Recopilacion datos ESVAL.xlsx',
'Temperatura', 'AB7:AB371');
        Tmax2011 = xlsread('Recopilacion datos ESVAL.xlsx',
'Temperatura', 'AG7:AG371');
        Tmax2012 = xlsread('Recopilacion datos ESVAL.xlsx',
'Temperatura', 'AL7:AL372');
        Tmax2013 = xlsread('Recopilacion datos ESVAL.xlsx',
'Temperatura', 'AQ7:AQ371');
        Tmax2014 = xlsread('Recopilacion datos ESVAL.xlsx',
'Temperatura', 'AV7:AV371');
        Tmax2015 = xlsread('Recopilacion datos ESVAL.xlsx',
'Temperatura', 'BA7:BA96');

        Tmax = vertcat(Tmax2005, Tmax2006, Tmax2007, Tmax2008,
Tmax2009, Tmax2010, Tmax2011, Tmax2012, Tmax2013, Tmax2014,
Tmax2015);

        Tmin2005 = xlsread('Recopilacion datos ESVAL.xlsx',
'Temperatura', 'B7:B371');
        Tmin2006 = xlsread('Recopilacion datos ESVAL.xlsx',
'Temperatura', 'G7:G371');
        Tmin2007 = xlsread('Recopilacion datos ESVAL.xlsx',
'Temperatura', 'L7:L371');
        Tmin2008 = xlsread('Recopilacion datos ESVAL.xlsx',
'Temperatura', 'Q7:Q372');
        Tmin2009 = xlsread('Recopilacion datos ESVAL.xlsx',
'Temperatura', 'V7:V371');
        Tmin2010 = xlsread('Recopilacion datos ESVAL.xlsx',
'Temperatura', 'AA7:AA371');
        Tmin2011 = xlsread('Recopilacion datos ESVAL.xlsx',
'Temperatura', 'AF7:AF371');
        Tmin2012 = xlsread('Recopilacion datos ESVAL.xlsx',
'Temperatura', 'AK7:AK372');
        Tmin2013 = xlsread('Recopilacion datos ESVAL.xlsx',
'Temperatura', 'AP7:AP371');
        Tmin2014 = xlsread('Recopilacion datos ESVAL.xlsx',
'Temperatura', 'AU7:AU371');
        Tmin2015 = xlsread('Recopilacion datos ESVAL.xlsx',
'Temperatura', 'AZ7:AZ96');

        Tmin = vertcat(Tmin2005, Tmin2006, Tmin2007, Tmin2008,
Tmin2009, Tmin2010, Tmin2011, Tmin2012, Tmin2013, Tmin2014,
Tmin2015);


        if ((argFrec == 'D') | (argFrec == 'd'))

            for i=1:length(ElSauce)
```

```
                LosAndesDDA(i) = ElSauce(i) + Bellavista(i) +
Miraflores(i);
            end
                LosAndesTmax = Tmax';
                LosAndesTmin = Tmin';

        else  % ((argFrec == 'M') | (argFrec == 'm'))

        for i=1:(round(length(ElSauce)/30)-1)
                DDAmensual = 0;
                TmaxMensual = 0;
                TminMensual = 0;
        for j=1:30
        DDAmensual = DDAmensual + ElSauce((i-1)*30+j) +
Bellavista((i-1)*30+j) + Miraflores((i-1)*30+j);
        TmaxMensual = TmaxMensual + Tmax((i-1)*30+j);
        TminMensual = TminMensual + Tmin((i-1)*30+j);
        end
            LosAndesDDA(i) = DDAmensual;
            LosAndesTmax(i) = TmaxMensual;
            LosAndesTmin(i) = TminMensual;
        end
        end % if

        Promedio(1) = mean2(LosAndesDDA);
        Promedio(2) = mean2(LosAndesTmax);
        Promedio(3) = mean2(LosAndesTmin);

        LosAndesDDA = LosAndesDDA./Promedio(1);

        LosAndesTmax = LosAndesTmax./Promedio(2);

        LosAndesTmin = LosAndesTmin./Promedio(3);

else %(temp == '0')
                % DEMANDA
        % El Sauce
        ElSauce2005 = xlsread('Recopilacion datos ESVAL.xlsx', 'El
Sauce', 'D7:D371');
        ElSauce2006 = xlsread('Recopilacion datos ESVAL.xlsx', 'El
Sauce', 'J7:J371');
        ElSauce2007 = xlsread('Recopilacion datos ESVAL.xlsx', 'El
Sauce', 'P7:P371');
        ElSauce2008 = xlsread('Recopilacion datos ESVAL.xlsx', 'El
Sauce', 'V7:V372');
        ElSauce2009 = xlsread('Recopilacion datos ESVAL.xlsx', 'El
Sauce', 'AB7:AB371');
        ElSauce2010 = xlsread('Recopilacion datos ESVAL.xlsx', 'El
Sauce', 'AH7:AH371');
        ElSauce2011 = xlsread('Recopilacion datos ESVAL.xlsx', 'El
Sauce', 'AN7:AN371');
        ElSauce2012 = xlsread('Recopilacion datos ESVAL.xlsx', 'El
Sauce', 'AT7:AT372');
```

```
        ElSauce2013 = xlsread('Recopilacion datos ESVAL.xlsx', 'El
Sauce', 'AZ7:AZ371');
        ElSauce2014 = xlsread('Recopilacion datos ESVAL.xlsx', 'El
Sauce', 'BF7:BF371');
        ElSauce2015 = xlsread('Recopilacion datos ESVAL.xlsx', 'El
Sauce', 'BL7:BL279');

        % Bellavista
        Bellavista2005 = xlsread('Recopilacion datos ESVAL.xlsx',
'Bellavista', 'D7:D371');
        Bellavista2006 = xlsread('Recopilacion datos ESVAL.xlsx',
'Bellavista', 'J7:J371');
        Bellavista2007 = xlsread('Recopilacion datos ESVAL.xlsx',
'Bellavista', 'P7:P371');
        Bellavista2008 = xlsread('Recopilacion datos ESVAL.xlsx',
'Bellavista', 'V7:V372');
        Bellavista2009 = xlsread('Recopilacion datos ESVAL.xlsx',
'Bellavista', 'AB7:AB371');
        Bellavista2010 = xlsread('Recopilacion datos ESVAL.xlsx',
'Bellavista', 'AH7:AH371');
        Bellavista2011 = xlsread('Recopilacion datos ESVAL.xlsx',
'Bellavista', 'AN7:AN371');
        Bellavista2012 = xlsread('Recopilacion datos ESVAL.xlsx',
'Bellavista', 'AT7:AT372');
        Bellavista2013 = xlsread('Recopilacion datos ESVAL.xlsx',
'Bellavista', 'AZ7:AZ371');
        Bellavista2014 = xlsread('Recopilacion datos ESVAL.xlsx',
'Bellavista', 'BF7:BF371');
        Bellavista2015 = xlsread('Recopilacion datos ESVAL.xlsx',
'Bellavista', 'BL7:BL279');

        % Miraflores
        Miraflores2005 = xlsread('Recopilacion datos ESVAL.xlsx',
'Miraflores', 'D7:D371');
        Miraflores2006 = xlsread('Recopilacion datos ESVAL.xlsx',
'Miraflores', 'J7:J371');
        Miraflores2007 = xlsread('Recopilacion datos ESVAL.xlsx',
'Miraflores', 'P7:P371');
        Miraflores2008 = xlsread('Recopilacion datos ESVAL.xlsx',
'Miraflores', 'V7:V372');
        Miraflores2009 = xlsread('Recopilacion datos ESVAL.xlsx',
'Miraflores', 'AB7:AB371');
        Miraflores2010 = xlsread('Recopilacion datos ESVAL.xlsx',
'Miraflores', 'AH7:AH371');
        Miraflores2011 = xlsread('Recopilacion datos ESVAL.xlsx',
'Miraflores', 'AN7:AN371');
        Miraflores2012 = xlsread('Recopilacion datos ESVAL.xlsx',
'Miraflores', 'AT7:AT372');
        Miraflores2013 = xlsread('Recopilacion datos ESVAL.xlsx',
'Miraflores', 'AZ7:AZ371');
        Miraflores2014 = xlsread('Recopilacion datos ESVAL.xlsx',
'Miraflores', 'BF7:BF371');
        Miraflores2015 = xlsread('Recopilacion datos ESVAL.xlsx',
'Miraflores', 'BL7:BL279');
```

```
        ElSauce = vertcat(ElSauce2005, ElSauce2006, ElSauce2007,
ElSauce2008, ElSauce2009, ElSauce2010, ElSauce2011, ElSauce2012,
ElSauce2013, ElSauce2014, ElSauce2015);
        Bellavista = vertcat(Bellavista2005, Bellavista2006,
Bellavista2007, Bellavista2008, Bellavista2009, Bellavista2010,
Bellavista2011, Bellavista2012, Bellavista2013, Bellavista2014,
Bellavista2015);
        Miraflores = vertcat(Miraflores2005, Miraflores2006,
Miraflores2007, Miraflores2008, Miraflores2009, Miraflores2010,
Miraflores2011, Miraflores2012, Miraflores2013, Miraflores2014,
Miraflores2015);


        if ((argFrec == 'D') | (argFrec == 'd')) % 2

            for i=1:length(ElSauce)
                LosAndesDDA(i) = ElSauce(i) + Bellavista(i) +
Miraflores(i);
            end

        else  % ((argFrec == 'M') | (argFrec == 'm'))

            for i=1:(round(length(ElSauce)/30)-1)
                DDAmensual = 0;
                for j=1:30
                DDAmensual = DDAmensual + ElSauce((i-1)*30+j) +
Bellavista((i-1)*30+j) + Miraflores((i-1)*30+j);
                end
                LosAndesDDA(i) = DDAmensual;
            end
        end % if 2

        Promedio(1) = mean2(LosAndesDDA);

        LosAndesDDA = LosAndesDDA./Promedio(1);
        LosAndesTmax = 0;
        LosAndesTmin = 0;

end % if 1
end % function
```

```
% FILTRACIÓN DE DATOS

function [LosAndesDDA_F, LosAndesTmax_F, LosAndesTmin_F] =
Filter(argDDA, argTmax, argTmin, n)

for i=1:(length(argDDA)-n+1)

    SLosAndesDDA_F = 0;
    SLosAndesTmax_F = 0;
    SLosAndesTmin_F = 0;

    for j=1:n
    SLosAndesDDA_F = SLosAndesDDA_F + argDDA(i+j-1);
    SLosAndesTmax_F = SLosAndesTmax_F + argTmax(i+j-1);
    SLosAndesTmin_F = SLosAndesTmin_F + argTmin(i+j-1);
    end

    LosAndesDDA_F(i) = SLosAndesDDA_F/n;
    LosAndesTmax_F(i) = SLosAndesTmax_F/n;
    LosAndesTmin_F(i) = SLosAndesTmin_F/n;
end


for k=2:(length(LosAndesDDA_F))
    DDA(k) = argDDA(k);
end


x = linspace(0, length(argDDA)-n+1, length(argDDA)-n+1);

plot(x, DDA, 'r.-')
hold on
plot(x, LosAndesDDA_F, 'b.-')

end
```

```
% OBTENCIÓN DE RED NEURONAL SIN RE-ENTRENAR

function [MSE, MAPE, MSPE, net] = NN_tMinMax_ED(argDDA, argTmax,
argTmin, argIn_dda, argIn_t, argHide, argNetKind, argLinearity,
argPhase)

if length(argDDA) < 1000
    P = 0.7;
else
    P = 0.8;
end

t = round(P*length(argDDA));
dda = argDDA(1:t);
dda_target = argDDA(t+1:length(argDDA));

if (argTmax ~= '0')
tmax = argTmax(1:t);
tmax_target = argTmax(t+1:length(argTmax));
end

if (argTmin ~= '0')
tmin = argTmin(1:t);
tmin_target = argTmin(t+1:length(argTmin));
end

d = argPhase;
in = argIn_dda;
hide = argHide;
out = 1;
inn = argIn_t;

%%%% 'D' Data

        for i=1:in
        delayDDA(i) = i-1;
        end

        for i=1:10
        delayTemp(i) = i-1;
        end

        % InDDA

        for i=1:(length(dda)-1-d)
            D_InDDA{1,i} = dda(i);
        end
        if(argTmax ~= '0')
        for i=1:(length(tmax)-1-d)
            D_InTmax{1,i} = tmax(i);
        end
        end
```

```
        if(argTmin ~= '0')
        for i=1:(length(tmin)-1-d)
            D_InTmin{1,i} = tmin(i);
        end
        end

        % Out
        for i=(2+d):(length(dda))
            D_Out{1,(i+1)-(2+d)} = dda(i);
        end

        % Target in
        for i=1:(length(dda_target)-1-d)
            D_TDDA{1,i} = dda_target(i);
        end
        if(argTmax ~= '0')
        for i=1:(length(tmax_target)-1-d)
            D_TTmax{1,i} = tmax_target(i);
        end
        end
        if(argTmin ~= '0')
        for i=1:(length(tmin_target)-1-d)
            D_TTmin{1,i} = tmin_target(i);
        end
        end

        % Target out
        for i=(2+d):(length(dda_target))
            D_RO{1,(i+1)-(2+d)} = dda_target(i);
        end

%%%% 'S' Data

        % In
        for i=1:(length(dda)-d-1-(in-1))
        for j=1:in
            S_InDDA(j,i) = dda(i+j-1);
        end
        end
        if(argTmax ~= '0')
        for i=1:(length(tmax)-d-1-(in-1))
        for j=1:inn
            S_InTmax(j,i) = tmax(i+j-1);
        end
        end
        end
        if(argTmin ~= '0')
        for i=1:(length(tmin)-d-1-(in-1))
        for j=1:inn
            S_InTmin(j,i) = tmin(i+j-1);
        end
        end
        end
```

```
        % Out
        for i=(1+in+d):(length(dda))
            S_Out(1,(i+1)-(1+in+d)) = dda(i);
        end

        % Target in
        for i=1:(length(dda_target)-d-1-(in-1))
        for j=1:in
            S_TDDA(j,i) = dda_target(i+j-1);
        end
        end
        if(argTmax ~= '0')
        for i=1:(length(tmax_target)-d-1-(in-1))
        for j=1:inn
            S_TTmax(j,i) = tmax_target(i+j-1);
        end
        end
        end
        if(argTmin ~= '0')
        for i=1:(length(tmin_target)-d-1-(in-1))
        for j=1:inn
            S_TTmin(j,i) = tmin_target(i+j-1);
        end
        end
        end

        % Target out
        for i=(1+in+d):(length(dda_target))
            S_RO(1,(i+1)-(1+in+d)) = dda_target(i);
        end


%%%%%%%%%%%%%%% Code NOT including temperatures %%%%%%%%%%%%%%%%%%

if (argTmax == '0' & argTmin == '0') % 1.1

    if (argNetKind == 'D') % 1.2
        In = D_InDDA;
        Out = D_Out;
        T = D_TDDA;
        RO = D_RO;

    elseif (argNetKind == 'S')
        In = S_InDDA;
        Out = S_Out;
        T = S_TDDA;
        RO = S_RO;

    end % 1.2 'D' or 'S' for data

    if (argNetKind == 'D' & argLinearity == 'L') % 1.3
```

```
        net = newfftd(In, Out, [1]);

        net.numInputs = 1;
        net.inputs{1}.size = 1;
        net.inputConnect = [1];
        net.biasConnect = [0];
        net.layerConnect = [0];
        net.outputConnect = [1];
        net.inputWeights{1,1}.delays = delayDDA;


    elseif (argNetKind == 'D' & argLinearity == 'N')
        net = newfftd(In, Out, [1], hide, {'logsig','purelin'});

        net.numInputs = 1;
        net.inputs{1}.size = 1;
        net.inputConnect = [1;0];
        net.biasConnect = [0;0];
        net.layerConnect = [0 0;1 0];
        net.outputConnect = [0 1];
        net.inputWeights{1,1}.delays = delayDDA;


    elseif (argNetKind == 'S' & argLinearity == 'L')
        net = newfftd(In, Out, [0]);

        net.numInputs = 1;
        net.inputs{1}.size = in;
        net.inputConnect = [1];
        net.biasConnect = [0];
        net.layerConnect = [0];
        net.outputConnect = [1];


    elseif (argNetKind == 'S' & argLinearity == 'N')
        net = newfftd(In, Out, [0], hide, {'logsig','purelin'});

        net.numInputs = 1;
        net.inputs{1}.size = in;
        net.inputConnect = [1;0];
        net.biasConnect = [0;0];
        net.layerConnect = [0 0;1 0];
        net.outputConnect = [0 1];

    end % 1.3 'D'&'N' or 'D'&'L' or 'S'&'N' or 'S'&'L'


%%%%%%%%%%%%%%% Code including temperatures %%%%%%%%%%%%%%%%%%

elseif ((argTmax == '0' & argTmin ~= '0') | (argTmax ~= '0' &
argTmin == '0'))
```

```
if (argNetKind == 'D') %2.1
    if (argTmax ~= '0')
    In = vertcat(D_InDDA, D_InTmax);
    elseif (argTmin ~= '0')
    In = vertcat(D_InDDA, D_InTmin);
    end
    Out = D_Out;
    T = vertcat(D_TDDA, D_TTmax);
    RO = D_RO;

elseif (argNetKind == 'S')
    if (argTmax ~= '0')
    In = vertcat(S_InDDA, S_InTmax);
    elseif (argTmin ~= '0')
    In = vertcat(S_InDDA, S_InTmin);
    end
    Out = S_Out;
    T = vertcat(S_TDDA, S_TTmax);
    RO = S_RO;

end % 2.1 'D' or 'S' for data

if (argNetKind == 'D' & argLinearity == 'L') % 2.2
    net = newfftd(In, Out, [1]);

    net.numInputs = 2;
    net.inputs{1}.size = 1;
    net.inputs{2}.size = 1;
    net.inputConnect = [1 1];
    net.biasConnect = [0];
    net.layerConnect = [0];
    net.outputConnect = [1];
    net.inputWeights{1,1}.delays = delayDDA;
    net.inputWeights{1,2}.delays = delayTemp;


elseif (argNetKind == 'D' & argLinearity == 'N')
    net = newfftd(In, Out, [1], hide, {'logsig','purelin'});

    net.numInputs = 2;
    net.inputs{1}.size = 1;
    net.inputs{2}.size = 1;
    net.inputConnect = [1 1;0 0];
    net.biasConnect = [0;0];
    net.layerConnect = [0 0;1 0];
    net.outputConnect = [0 1];
    net.inputWeights{1,1}.delays = delayDDA;
    net.inputWeights{1,2}.delays = delayTemp;


elseif (argNetKind == 'S' & argLinearity == 'L')
```

```
        net = newfftd(In, Out, [0]);

        net.numInputs = 2;
        net.inputs{1}.size = in;
        net.inputs{2}.size = inn;
        net.inputConnect = [1 1];
        net.biasConnect = [0];
        net.layerConnect = [0];
        net.outputConnect = [1];


    elseif (argNetKind == 'S' & argLinearity == 'N')
        net = newfftd(In, Out, [0], hide, {'logsig','purelin'});

        net.numInputs = 2;
        net.inputs{1}.size = in;
        net.inputs{2}.size = inn;
        net.inputConnect = [1 1;0 0];
        net.biasConnect = [0;0];
        net.layerConnect = [0 0;1 0];
        net.outputConnect = [0 1];

    end % 2.2 'D'&'N' or 'D'&'L' or 'S'&'N' or 'S'&'L'


%%%%%%%%%%%%%%% Code including both Temperatures %%%%%%%%%%%%%

elseif (argTmax ~= '0' & argTmin ~= '0' & argDDA ~= '0')

    if (argNetKind == 'D') % 3.1
        In = vertcat(D_InDDA, D_InTmax, D_InTmin);
        Out = D_Out;
        T = vertcat(D_TDDA, D_TTmax, D_TTmin);
        RO = D_RO;

    elseif (argNetKind == 'S')
        In = vertcat(S_InDDA, S_InTmax, S_InTmin);
        Out = S_Out;
        T = vertcat(S_TDDA, S_TTmax, S_TTmin);
        RO = S_RO;

    end % 3.1 'D' or 'S' for data

     if (argNetKind == 'D' & argLinearity == 'L') % 3.2
        net = newfftd(In, Out, [1]);

        net.numInputs = 3;
        net.inputs{1}.size = 1;
        net.inputs{2}.size = 1;
        net.inputs{3}.size = 1;
        net.inputConnect = [1 1 1];
        net.biasConnect = [0];
```

```
        net.layerConnect = [0];
        net.outputConnect = [1];
        net.inputWeights{1,1}.delays = delayDDA;
        net.inputWeights{1,2}.delays = delayTemp;
        net.inputWeights{1,3}.delays = delayTemp;


    elseif (argNetKind == 'D' & argLinearity == 'N')
        net = newfftd(In, Out, [1], hide, {'logsig','purelin'});

        net.numInputs = 3;
        net.inputs{1}.size = 1;
        net.inputs{2}.size = 1;
        net.inputs{3}.size = 1;
        net.inputConnect = [1 1 1;0 0 0];
        net.biasConnect = [0;0];
        net.layerConnect = [0 0;1 0];
        net.outputConnect = [0 1];
        net.inputWeights{1,1}.delays = delayDDA;
        net.inputWeights{1,2}.delays = delayTemp;
        net.inputWeights{1,3}.delays = delayTemp;


    elseif (argNetKind == 'S' & argLinearity == 'L')
        net = newfftd(In, Out, [0]);

        net.numInputs = 3;
        net.inputs{1}.size = in;
        net.inputs{2}.size = inn;
        net.inputs{3}.size = inn;
        net.inputConnect = [1 1 1];
        net.biasConnect = [0];
        net.layerConnect = [0];
        net.outputConnect = [1];


    elseif (argNetKind == 'S' & argLinearity == 'N')
        net = newfftd(In, Out, [0], hide, {'logsig','purelin'});

        net.numInputs = 3;
        net.inputs{1}.size = in;
        net.inputs{2}.size = inn;
        net.inputs{3}.size = inn;
        net.inputConnect = [1 1 1;0 0 0];
        net.biasConnect = [0;0];
        net.layerConnect = [0 0;1 0];
        net.outputConnect = [0 1];

    end % 3.2 'D'&'N' or 'D'&'L' or 'S'&'N' or 'S'&'L'

end % 1.1 argTmax = '0' or 'LosAndesTmax'
```

```
 % Initialization
    net = init(net);

    % Train
    ts=.1;
    epoca = 100;
    error = 1e-10;
    lr = 0.01;
    gradiente = 0;
    mu = 1;
    mu_dec = 0.8;
    mu_inc = 1.2;

    % Training parameters
    net.trainParam.epochs = epoca;
    net.trainParam.goal = error;
    net.trainParam.min_grad = gradiente;
    net.trainParam.alpha = lr;
    net.trainParam.mu = mu;
    net.trainParam.mu_dec = mu_dec;
    net.trainParam.mu_inc = mu_inc;
    net.trainParam.max_fail = 80;

    net = train(net, In, Out);

    % Simulation
    y = sim(net,T);

    if (argNetKind == 'D') % 4.1
        MSE = mse(y,RO)
        squareMape = cellfun(@(x,y) (abs((y-x)/x)),RO,y);
        MAPE = sum(squareMape)/length(RO)*100
        squareMspe = cellfun(@(x,y) ((abs((y-x)/x))^2),RO,y);
        MSPE = sum(squareMspe)/length(RO)*100

        x = linspace(0, length(y), length(y));
        plot( x, cell2mat(RO), 'r.-', x, cell2mat(y), 'b.-')

    elseif (argNetKind == 'S')
        MSE = mse(y,RO);
        MAPE = 100/length(RO)*sum((abs((y-RO)./RO))); %.^2);
        MSPE = sum((abs((y-RO)./RO)).^2)/length(RO)*100;

        x = linspace(0, length(y), length(y));
        plot( x, RO, 'r.-', x, y, 'b.-')

    end % 4.1 'D' or 'S' for mse/mspe

end % function
```

```
% RE-ENTRENAMIENTO

function [MSE, MAPE, MSPE, newnet] = ReTrain(argDDA, argTmax,
argTmin, argFrec, argPhase, Promedio)

e = argDDA;
tmax = argTmax;
tmin = argTmin;
t = round(0.2*length(argDDA));
d = argPhase;

if (argFrec == 'd') %1

    if (d == 0) %1.1
    load Snet_0d;
    net = Snet_0d;

    elseif (d == 30)
    load Snet_30d;
    net = Snet_30d;

    elseif (d == 60)
    load Snet_60d;
    net = Snet_60d;

    elseif (d == 90)
    load Snet_90d;
    net = Snet_90d;

    elseif (d == 180)
    load Snet_180d;
    net = Snet_180d;

    elseif (d == 270)
    load Snet_270d;
    net = Snet_270d;

    elseif (d == 360)
    load Snet_360d;
    net = Snet_360d;
    end  %1.1

elseif (argFrec == 'df')

    if (d == 0) %1.2
    load F_Snet_0d;
    net = F_Snet_0d;

    elseif (d == 15)
    load F_Snet_15d;
    net = F_Snet_15d;
```

```
    elseif (d == 30)
    load F_Snet_30d;
    net = F_Snet_30d;

    elseif (d == 45)
    load F_Snet_45d;
    net = F_Snet_45d;

    elseif (d == 60)
    load F_Snet_60d;
    net = F_Snet_60d;

    elseif (d == 90)
    load F_Snet_90d;
    net = F_Snet_90d;

    elseif (d == 180)
    load F_Snet_180d;
    net = F_Snet_180d;

    elseif (d == 270)
    load F_Snet_270d;
    net = F_Snet_270d;

    elseif (d == 360)
    load F_Snet_360d;
    net = F_Snet_360d;

    end  %1.2


else  %  (argFrec == 'm')

    if (d == 0) %1.3
    load Snet_0m;
    net = Snet_0m;

    elseif (d == 1)
    load Snet_1m;
    net = Snet_1m;

    elseif (d == 2)
    load Snet_2m;
    net = Snet_2m;

    elseif (d == 3)
    load Snet_3m;
    net = Snet_3m;

    elseif (d == 6)
    load Snet_6m;
    net = Snet_6m;
```

```
    elseif (d == 9)
    load Snet_9m;
    net = Snet_9m;

    elseif (d == 12)
    load Snet_12m;
    net = Snet_12m;

    end %1.3

end %1

% Train
    ts=.1;                  % tiempo de sample
    epoca = 100;            % epocas
    error = 1e-10;          % error global
    lr = 0.01;              % razon de aprendizaje
    gradiente = 0;          % minimo gradiente
    mu = 1;                 % cambio este parametro para que no...
    mu_dec = 0.8;           % ...converger tan rapido
    mu_inc = 1.2;

    % Training parameters
    net.trainParam.epochs = epoca;
%     net.trainParam.goal = error;
    net.trainParam.min_grad = gradiente;
    net.trainParam.alpha = lr;
    net.trainParam.mu = mu;
    net.trainParam.mu_dec = mu_dec;
    net.trainParam.mu_inc = mu_inc;
    net.trainParam.max_fail = 80;


    [T, RO, In, Out] = Target(e, tmax, tmin, d, net.inputs{1}.size,
net.inputs{2}.size, argFrec);
    newnet = train(net, In, Out);
    y = sim(newnet,T);

        MSE = mse(y,RO);
        MAPE = 100/length(RO)*sum((abs((y-RO)./RO)));
        MSPE = sum((abs((y-RO)./RO)).^2)/length(RO)*100;

        x = linspace(0, length(y*Promedio(1)),
length(y*Promedio(1)));
        plot( x, RO*Promedio(1), 'r.-', x, y*Promedio(1), 'b.-')

end % function
```