ESCUELA SUPERIOR POLITÉCNICA DE CHIMBORAZO

ANÁLISIS DEL RENDIMIENTO DE ALGORITMOS DE ENTRENAMIENTO DE REDES NEURONALES ARTIFICIALES, APLICADAS AL MODELAMIENTO DINÁMICO DE REPRESAS HIDROELÉCTRICAS, MEDIANTE EL ERROR DE PREDICCIÓN DEL NIVEL DE EMBALSE DE AGUA

EDISON XAVIER CHAFLA YAMBAY

Trabajo de Titulación modalidad: Proyecto de Investigación y Desarrollo, presentado ante el Instituto de Posgrado y Educación Continua de la ESPOCH, como requisito para la obtención del grado de:

MAGISTER EN SISTEMAS DE CONTROL Y AUTOMATIZACIÓN INDUSTRIAL

RIOBAMBA – ECUADOR

Febrero 2019

ESCUELA SUPERIOR POLITECNICA DE CHIMBORAZO

CERTIFICACIÓN

EL TRIBUNAL DEL TRABAJO DE TITULACIÓN CERTIFICA QUE:

El trabajo de Titulación modalidad Proyectos de Investigación y Desarrollo, denominado "ANÁLISIS DEL RENDIMIENTO DE ALGORITMOS DE ENTRENAMIENTO DE REDES NEURONALES ARTIFICIALES, APLICADAS AL MODELAMIENTO DINÁMICO DE REPRESAS HIDROELÉCTRICAS, MEDIANTE EL ERROR DE PREDICCIÓN DEL NIVEL DE EMBALSE DE AGUA", de responsabilidad del Ingeniero Edison Xavier Chafla Yambay, ha sido prolijamente revisado y se autoriza su presentación.

Lic. Pepita Alarcón Parra, M.Sc.	----------
PRESIDENTE	**FIRMA**
Ing. Jorge Hernández Ambato, Ph.D.	----------
DIRECTOR	**FIRMA**
Ing. Gabriel Asqui Santillan, M.Sc.	----------
MIEMBRO	**FIRMA**
Ing. Jorge Paucar Samaniego, M. Sc.	----------
MIEMBRO	**FIRMA**

Riobamba, Febrero del 2019

DERECHOS INTELECTUALES

Yo, Edison Xavier Chafla Yambay, declaro que soy responsable de las ideas doctrinas y resultados expuestos en el **Trabajo de Titulación modalidad Proyectos de Investigación y Desarrollo**, y que el patrimonio intelectual generado por la misma pertenece exclusivamente a la Escuela Superior Politécnica de Chimborazo.

Ing. Edison Xavier Chafla Yambay.

Cc.: 0602616278

DEDICATORIA

A Dios, a mis padres, Luis Gonzalo (+) e Inés Aurora, gracias por las alas, las raíces, por la vida.

A ti, Paola con amor.

Edison X. Chafla Y.

AGRADECIMIENTO

A la Escuela Superior Politécnica de Chimborazo, Instituto de Posgrado y Educación Continua, para la realización del presente tema de tesis.

A mis familiares en especial a mi madre mi eterna gratitud por su apoyo incondicional en cada una de las etapas de mi vida.

A mis amigos de manera particular al compañero de aulas y amigo Ing. Gabriel Asqui, por el acompañamiento en este proyecto investigativo.

Edison X. Chafla Y.

CONTENIDO

CAPÍTULO IV

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

ÍNDICE DE GRÁFICOS

RESUMEN

El objetivo de esta investigación fue evaluar los algoritmos de entrenamiento de la librería Keras de Python, aplicadas al modelamiento dinámico de represas hidroeléctricas, mediante el error de predicción del nivel de embalse de agua. Para esto se utilizó datos históricos (2005-2016) del nivel, caudal y potencia activa de la Central Hidroeléctrica Agoyán, almacenados en el gestor de base de datos PostgredSQL, a los cuales se los dividió en temporadas climáticas y se los procesó utilizando técnicas de estacionamiento de señales y de normalización. Para el entrenamiento de los modelos se desarrolló una plataforma de software de Python, con el uso de los algoritmos de entrenamiento de la librería Keras más el back-end de Tensorflow. El procesador utilizado para estas tareas fue una unidad de procesamiento gráfico GPU Nvidia 1050Ti. A través del análisis de la varianza ANOVA, se obtuvo una probabilidad (p) de $6{,}02157\text{E}-44$ y $1.4024\text{E}-42$ para el error de entrenamiento y error de validación respectivamente, lo cual descartó la hipótesis nula ya que en ambos caso la probabilidad fue menor a 0.05 es decir ($p < n$) para un nivel de confianza (n) del 95%, la prueba de Tukey determinó que el algoritmo Nadam tiene la menor diferencia significativa respecto al resto, la eficiencia estadística comprobó que el algoritmo Nadam es el más eficiente $Var(Nadam) < Var(SGD)$. El modelo entrenado con el algoritmo Nadam alcanzó un predictor de nivel efectivo hasta un umbral de 48 horas, consiguiendo un RMSE mínimo de 0.035876 [m] y un máximo de 0.344913 [m] del error de nivel de agua de embalse de la presa. En este caso de estudio se utilizó el backend de TensorFlow, pero existe actualmente otros backends como: Theano y CNTK, se recomienda entrenar los algoritmos en estas estructuras y probar su rendimiento.

PALABRAS CLAVES: <TECNOLOGÍA Y CIENCIAS DE LA INGENIERÍA>, <CONTROL AUTOMÁTICO>, <INTELIGENCIA ARTIFICIAL>, <REDES NEURONALES ARTIFICIALES (RNA)>, <PREDICTOR>, <DEEP LEARNING>, <PYTHON (SOFTWARE)>, <KERAS(SOFTWARE)>

ABSTRACT

The objective of this investigation was to evaluate training algorithms from the Keras Python library; applying to the dynamic modelling of hydroelectric dams, by means of the prediction error of the water reservoir level. Historical dates were used (2005-2016) such as level, row and active power from the Agoyan Hydroelectric Power Plant stored in the database manager PostgreSQL which are divided into climatic seasons and are processed using signalling and standardization techniques. For the training of the models, a Python software platform was developed, with the use of the training algorithms of the Keras library plus the Tensorflow back-end. The processor used for these tasks was graphics processing unit GPU Nvidia 1050Ti. Through the variance analyze ANOVA a probability obtained was (p) of 6,02157E - 44 and 1.4024E - 42 for training error and validation error respectively, which rule out the null hypothesis since in both cases the probability was less than 0.05 this means $(p < n)$ for a confidence level (n) of 95% the Tukey test determine that Nadam algorithm It has the least significant difference from other statistical efficiency found that the Nadam algorithm is the most efficient $Var(Nadam) < Var(SGD)$. The model with training the algorithm Nadam reached a predictor of an effective level to a threshold 48 hours obtaining an RMSE minimum of 0.035876 [m] and a maximum of 0.344913 [m] of reservoir water level error of the dam. In this case study the backend Tensorflow was used but there are currently others like Theano and CNTK, it is recommended to train the algorithms in these structures and test their performance.

KEY WORDS: <ENGINEERING SCIENCE AND TECHNOLOGY>, <AUTOMATIC CONTROL>, <ARTIFICIAL INTELLIGENCE>, <ARTIFICIAL NEURAL NETWORK (ANN)>, <PREDICTOR>, <DEEP LEARNING>, <PYTHON (SOFTWARE)>, <KERAS (SOFTWARE)>.

CAPÍTULO I

1. INTRODUCCIÓN

En este capítulo se detalla el problema de investigación, el planteamiento y formulación, así como también la justificación, los objetivos generales, específicos e hipótesis de la investigación.

1.1. Planteamiento del problema

La energía es fundamental para el crecimiento económico y la sostenibilidad ambiental, y se ha descrito como "el hilo " que une el crecimiento económico, la equidad social y la sostenibilidad ambiental. La producción y consumo de electricidad de un país son indicadores básicos de su tamaño y nivel de desarrollo. Cifras publicadas por el Banco Mundial, en el 2017, 1060 millones de personas aún viven sin electricidad, lo que representa tan solo una leve mejora desde 2012. Según esto, el mundo alcanzaría el 92 % de electrificación para el 2030 (BANCO MUNDIAL, 2017).

El consumo de energía eléctrica (kWh per cápita), que es la relación que existe entre el consumo de energía eléctrica y la población; de acuerdo al Banco Mundial, para el año 2014 se ubicó en 3.126,3 kWh/hab a nivel mundial (BANCO MUNDIAL, 2017). En el Ecuador este índice para el año 2016 fue de 1.143,31 kWh/hab, con un consumo de 18.897,43 GWh, para una población de 16`528.730 habitantes, publicado por la Agencia de Regulación y Control de Electricidad (ARCONEL), y según datos del Instituto Ecuatoriano de Estadísticas y Censos (INEC)(Galarza, 2017)(INEC, 2017).

Teniendo en cuenta esta problemática el Plan Maestro de Electrificación 2013-2022, emitido por Consejo Nacional de Electricidad (CONELEC), dentro del Capítulo 1 referente a los aspectos de sostenibilidad social y ambiental, indica: “La eficiencia energética, (…), trata sobre el conjunto de acciones, en ejecución y planificadas, tendientes a optimizar los recursos energéticos renovables y consumir la menor cantidad posible de energía, (…), y con el menor impacto sobre el medio ambiente” (CONELEC, 2013). Por ende, la optimización de recursos energéticos se convierte en una prioridad.

Actualmente el Ecuador a través del MEER se encuentra ejecutando proyectos para la generación de energía renovable, como son: Coca Codo Sinclair, Minas San Francisco, Delsitanisagua, Manduriacu, Masar Dudas, Toachi Pilatón, Quijos, Sopladora y Villonaco (Galarza, 2017).

En este sentido, de acuerdo a políticas de Estado referente a la eficiencia energética, y tomando en cuenta que la proyección del índice de población del país para el año 2020 sería de 17´512.663 habitantes (INEC, 2017), es imperativo la implementación de centrales hidroeléctricas que aseguren el abastecimiento normal de energía eléctrica para la población. Sin embargo, un punto muy importante es la optimización en la producción de la energía en las centrales en operación, como el caso de estudio realizado por G. Asqui, sobre la "Predicción del nivel de agua del embalse, basado en redes neuronales, para la mejora de la planificación de producción de energía en la central hidroeléctrica Agoyán" (Asqui, 2017).

G. Asqui y J. Hernández utilizaron el algoritmo de entrenamiento Broyden–Fletcher–Goldfarb–Shanno (BFGS) para la Red Neuronal Artificial (RNA) perceptrón multicapa, con el objeto de obtener un predictor de nivel basado en dos RNAs, una utilizada en el modelamiento dinámico de la presa y otra para predecir el caudal, sin embargo el resultado total del sistema depende de la combinación del error de estas dos RNAs (Asqui, 2017, p. 20).

El presente trabajo pretende estudiar el comportamiento del error del modelo de una presa hidroeléctrica, sabiendo que si se logra disminuir el error del modelo también se disminuirá el error del predictor del nivel en su conjunto, con este fin se pretende comparar los distintos algoritmos de entrenamiento de RNA utilizando una Unidad de Procesamiento Gráfico (GPU) y verificando su influencia positiva o negativa en el error con el fin de validar el algoritmo de entrenamiento más adecuado para el modelamiento dinámico de la presa y así conseguir mejorar los resultados de predicción.

1.2. Formulación del problema

El rendimiento de los algoritmos de entrenamiento de redes neuronales artificiales de la librería Keras, determinado mediante el análisis de los errores de entrenamiento y validación obtenidos durante el modelamiento dinámico de centrales hidroeléctricas, permitirá identificar el algoritmo más y menos eficiente en la predicción del nivel de embalse de agua de la represa.

1.3. Preguntas directrices

¿Cuáles son los algoritmos de entrenamiento de RNAs existentes dentro de la herramienta Open Source "Keras" que pueden ser utilizados para el modelamiento dinámico de represas hidroeléctricas?

¿Una GPU podrá ser utilizada para evaluar el desempeño de los algoritmos de entrenamiento de RNAs proporcionados por la herramienta "¿Open Source Keras", mediante su aplicación al modelamiento dinámico de la represa hidroeléctrica Agoyán?

¿El error de predicción de nivel de embalse de agua de una represa hidroeléctrica podría ser afectado por la utilización de algoritmos de entrenamiento de RNAs más y menos eficientes?

1.4. Justificación del problema

La demanda mundial de energía primaria ha crecido a un promedio anual del 1,8% desde 2011. A partir del año 2015, la energía renovable proyectó un estimado del 19.3% de consumo de energía global. De este porcentaje, la participación de la energía hidroeléctrica representó el 3,6 %. En el año 2016, el Ecuador ocupó el tercer lugar a nivel mundial por la capacidad hidroeléctrica instalada según el Global Status Report Renewables, 2017, como se muestra en la figura 1-1 (REN21, 2017). Esto se debe a la implementación de proyectos como: Coca Codo Sinclair, que entrega 1.5 GW y la planta Sopladora de 487 MW, las cuales reúnen casi la mitad de las necesidades de electricidad del país.

TOP FIVE COUNTRIES

Annual Investment / Net Capacity Additions / Production in 2016

	1	2	3	4	5
Investment in renewable power and fuels (not including hydro > 50 MW)	**China**	United States	United Kingdom	Japan	Germany
Investment in renewable power and fuels per unit GDP[1]	**Bolivia**	Senegal	Jordan	Honduras	Iceland
Geothermal power capacity	**Indonesia**	Turkey	Kenya	Mexico	Japan
Hydropower capacity	**China**	Brazil	Ecuador	Ethopia	Vietnam
Solar PV capacity	**China**	United States	Japan	India	United Kingdom
Concentrating solar thermal power (CSP) capacity[2]	**South Africa**	China	–	–	–
Wind power capacity	**China**	United States	Germany	India	Brazil
Solar water heating capacity	**China**	Turkey	Brazil	India	United States
Biodiesel production	**United States**	Brazil	Argentina/Germany/Indonesia		
Fuel ethanol production	**United States**	Brazil	China	Canada	Thailand

Figura 1-1: Capacidad hidroeléctrica instalada.
Fuente: ("Global Status Report Renewables", 2017)

De acuerdo al Plan Nacional del Buen Vivir, en su Objetivo 7.7, es prioridad del Estado, el promover la eficiencia y una mayor participación de energías renovables sostenibles como medida de prevención de la contaminación ambiental (SENPLADES, 2013).

Además en el literal a) del objetivo 7.7, se menciona que es necesario implementar tecnologías, infraestructuras y esquemas tarifarios, para promover el ahorro y eficiencia energética en los diferentes sectores de la economía(SENPLADES, 2013). Este es el caso de la Central Hidroeléctrica Agoyán, en la cual se han realizado estudios como el elaborado por G. Asqui y J. Hernández, que pretenden mediante la predicción del nivel de agua del embalse, basado en redes neuronales, mejorar la planificación de producción de energía (Asqui, 2017, p. 63).

El estudio elaborado por G. Asqui y J. Hernández fue sometido a prueba por 22 días y se obtuvo un error RMSE (Root Square Error) promedio de 9.96 metros cúbicos por segundo, con un porcentaje de aciertos de 45.45%, para el predictor de caudal. Por otro lado, la predicción del nivel de embalse obtuvo un RMSE promedio de 0.2735 m. s. n. m, con un porcentaje de aciertos de 86.36%. Dichos estudios se realizaron manteniendo un umbral de ocho (8) horas de predicción. Se concluyó que dicho umbral alcanzado no afecta aún los índices de producción (Asqui, 2017).

El presente trabajo de investigación pretende someter a evaluación a los diferentes algoritmos de entrenamiento de RNAs, para verificar cuál es su influencia en el error del modelo de la presa tomando en cuenta que en el anterior estudio únicamente se validó con un solo algoritmo de entrenamiento utilizando CPU (Central Process Unit). De esta manera conseguir optimizar el uso del recurso agua y lograr producir energía eléctrica en la Hidroeléctrica Agoyán de una manera más eficiente.

Para realizar el estudio de los distintos algoritmos de entrenamiento de las RNA es necesaria la aplicación de herramientas informáticas que permitan la aplicación de los distintos modelos. En este sentido y de acuerdo al Decreto Presidencial 1014 expedido el 10 de abril de 2008, referente al uso de software libre, menciona en su artículo 1: "Establecer como política pública para las Entidades de la Administración Pública Central la utilización de Software Libre en sus sistemas y equipamientos informáticos" (Correa, 2010). Por lo que acogiéndose a este decreto se implementará una plataforma Open Source basada en GPU

Con el objetivo de reducir los costos de la implementación de esta investigación es necesario la utilización de una herramienta informática basada en software libre. En particular, se pretende utilizar programación en lenguaje Python para implementar algoritmos de entrenamiento de redes neuronales artificiales basados en la librería Keras y el back-end de Tensorflow. Esto es importante considerar ya que actualmente el software licenciado es costoso, por lo cual es imperiosa la necesidad de optar herramientas basadas en software libre.

1.5. Objetivos

1.5.1. Objetivo general

Analizar el rendimiento de los algoritmos de entrenamiento de redes neuronales artificiales de la librería Keras, mediante los errores de entrenamiento y validación obtenidos durante el modelamiento dinámico de centrales hidroeléctricas, para identificar el algoritmo más y menos eficiente en la predicción del nivel de embalse de agua.

1.5.2. Objetivos específicos

- Identificar los algoritmos de entrenamiento de RNAs existentes dentro de la herramienta Open Source Keras que pueden ser utilizados para el modelamiento dinámico de represas hidroeléctricas mediante investigación bibliográfica.

- Implementar una plataforma de evaluación para los algoritmos de entrenamiento de RNAs de la herramienta Open Source Keras aplicadas al modelamiento dinámico de la represa Agoyán usando GPU.

- Evaluar el rendimiento de los algoritmos de entrenamiento de RNAs más y menos eficientes de la herramienta Open Source Keras durante la predicción del nivel de embalse de agua de la represa hidroeléctrica Agoyán.

1.6. Hipótesis

1.6.1. Hipótesis general

Los errores de entrenamiento y validación obtenidos por los algoritmos de entrenamiento de redes neuronales artificiales de la librería Keras, aplicadas al modelamiento dinámico de represas hidroeléctricas, permiten identificar el algoritmo más y menos eficiente para predicción del nivel de embalse de agua de la represa.

1.6.2. Hipótesis Específicas

- Al menos un algoritmo de entrenamiento de RNAs existentes dentro de la herramienta Open Source Keras pueden ser utilizados para el modelamiento dinámico de represas hidroeléctricas.

- Una Unidad de Procesamiento Gráfico (GPU) sirve como plataforma Hardware para la evaluación de algoritmos de entrenamiento de RNAs de la herramienta Open Source Keras.

- El error de predicción de nivel de agua de una represa hidroeléctrica es atenuado por el algoritmo de entrenamiento de RNAs de la librería Open Source Keras más eficiente.

CAPÍTULO II

2. MARCO TEÓRICO

2.1 Estado del Arte

2.2.1. Antecedentes

Los sistemas dinámicos son un área joven de las matemáticas, aunque se remontan a Newton con sus estudios de la mecánica celeste, y a Henry Poincaré, quién inició el estudio cualitativo de las ecuaciones diferenciales. Sin embargo, fue hace apenas unos 40 años que los sistemas dinámicos se establecieron como un área propiamente dicha, gracias al trabajo destacado de matemáticos e ingenieros como: Smale, Arnold, Lyapunov.

"El concepto de sistemas dinámicos, se podría decir que se trata del estudio de sistemas deterministas, es decir, se consideran situaciones que dependan de algún parámetro dado, que frecuentemente sea el tiempo, y que varían de acuerdo con leyes establecidas." (Ortega, 2000). De manera que el conocimiento de la situación en un momento dado permite reconstruir el pasado y predecir el futuro

Muchas veces se escucha hablar sobre la inteligencia artificial, pero ¿Qué es realmente? A lo largo de la historia son numerosas las definiciones que se han dado sobre este tema; algunas de ellas son: Capacidad que tienen las máquinas para realizar tareas que en el momento son realizadas por seres humanos. "Rama de la ciencia de la computación que estudia resolución de problemas no algorítmicos mediante el uso de cualquier técnica de computación, sin tener en cuenta la forma de razonamiento subyacente a los métodos que se apliquen para lograr esa resolución" (Galán & Martínez, 2010).

Las redes neuronales artificiales (RNA), son una aproximación a la inteligencia artificial fundamentada en el cerebro de los mamíferos, más explícitamente en las neuronas biológicas. En la actualidad son ampliamente usadas en diferentes campos del conocimiento para: análisis de datos, reconocimiento de patrones y problemas de predicción (Isasi & Galván León, 2004).

2.2.2. Bases teóricas.

M. Ghiassi, H. Saidane, D. Zimbra, en el estudio denominado, Modelo de una red neuronal artificial dinámica para pronosticar eventos de series de tiempo, utilizan una arquitectura diferente a los modelos tradicionales, para evaluar la efectividad del método, pronosticaron una serie de puntos de referencia estándar en la investigación de series temporales de la literatura de pronósticos, sus resultados mostraron que este método es más preciso y tiene un rendimiento significativo en comparación a los modelos de red neuronal tradicional y media móvil autoregresiva integrada (ARIMA) (Ghiassi, Saidane, & Zimbra, 2005).

G. Zhang, B. Patuwo, M. Hu, en el trabajo de investigación denominado, Un estudio de simulación de redes neuronales artificiales para pronóstico de series temporales no lineales, mediante la simulación de los factores principales (nodos de entrada, nodos ocultos y tamaño de la muestra) de una red neuronal artificial. De acuerdo con los resultados muestran que las RNAs son herramientas valiosas para modelar y pronosticar series temporales no lineales, mientras que los métodos tradicionales no son tan competentes para esta área, además indican que la cantidad de nodos de entrada es mucho más importante que la cantidad de nodos ocultos en la creación de modelos de redes neuronales para la predicción (Zhang, Patuwo, & Hu, 2001).

H. Raman, N. Sunilkumar, en el estudio de; Modelación multivariante de series temporales de recursos hídricos utilizando redes neuronales artificiales, en el cual indican que, la mayoría de los procedimientos de modelado de series de tiempo caen dentro del modelo de promedio autorregresivo multivariante (ARMA). Este artículo investigó el uso de redes neuronales artificiales en el campo de la generación de flujo sintético y lo comparó con el método estadístico, aplicado a registros mensuales de flujo de entrada para dos sitios de yacimiento (Raman & Sunilkumar, 1995).

R. Taormina, K. Chau, R. Sethi, en su trabajo de investigación denominado, Simulación de redes neuronales artificiales de niveles de agua subterránea por hora en un sistema acuífero costero de la laguna de Venecia, para lograr este propósito el modelo calibra primero el conjunto de datos de entrenamiento para realizar predicciones de 1 hora por adelantado de los niveles futuros de agua subterránea utilizando los niveles de agua subterránea observados en el pasado y las entradas externas. Luego se simuló mediante un conjunto de datos alimentado iterativamente los niveles de agua subterránea pronosticada junto con datos externos reales (Taormina, Chau, & Sethi, 2012).

M. Ghorbani, R Khatibi, A. Aytek, en su trabajo investigativo, Predicción de nivel de agua de mar utilizando Programación Genética (PG) y comparando con redes neuronales artificiales (RNAs), para el cual se utilizó varios intervalos de tiempo usando registro de series de tiempo pasadas de diciembre 1991 a diciembre de 2002, en intervalos de tiempo de 12h, 24h, 5 días y 10 días, utilizando los niveles del mar observados. Se comparó un conjunto de resultados proporcionados por la PG con resultados de un modelo de RNA, determinaron que ambas tecnologías GP y RNAs pueden considerarse como alternativas de análisis armónico (Ghorbani, Khatibi, & Aytek, 2010).

F. Chang, Y. Chang, en el estudio denominado, Sistema adaptativo de inferencia neruo-difusa para la predicción del nivel del agua de embalse, utilizaron un Sistema Adaptativo de Interferencia Neruo- Difusa (ANFIS), aplicado en la presa de Shihmen en Taiwan, utilizaron un gran número de (132) eventos de tifones y lluvias fuertes con 8640 conjuntos de datos por hora recopilados en los últimos 31 años. Desarrollaron dos modelos ANFIS, uno con decisión humana como entrada y otro sin él. Los resultados demuestran que el ANFIS se puede aplicar con éxito y proporcionar alta precisión y confiabilidad para el pronóstico del nivel de agua de la presa en las próximas tres horas (Chang & Chang, 2006).

A. Zuñiga, en su trabajo de investigación, denominado Pronóstico de caudales medios mensuales empleando sistemas Neurofuzzy, en el cual utiliza una metodología de Redes Neuro-Fuzzy, específicamente el modelo ANFIS (Adaptative Neuro Based Fuzzy Inference System), aplicada al problema del pronóstico de caudales afluentes mensuales en la central hidroeléctrica Paute y Daule Peripa (Zuñiga & Jórdan, 2005).

Estudios basados predicción de caudales de ríos utilizados en centrales hidroeléctricas mediante la implementación de redes neuronales artificiales o lógica difusa como herramientas de análisis, han sido muy escasos, como por ejemplo el desarrollado por G. Asqui, en su trabajo de titulación denominado Predicción del nivel de agua del embalse, basado en redes neuronales, para la mejora de la planificación de producción de energía en la central hidroeléctrica Agoyán, aplica el algoritmo de entrenamiento Broyden-Fletcher-Goldfard-Shanno (BFGS), con una plataforma de software con el lenguaje Python (Asqui, 2017).

2.2. Series temporales

Las series temporales se puede definir como una sucesión de observaciones de una variable tomados en varios instantes de tiempo. Se caracterizan porque su evolución temporal no depende específicamente del tiempo, sino de los valores de la serie en instantes de tiempo. Son utilizadas

para la predicción o interpretación del comportamiento de una variable en el futuro (extrapolación pronostica) o en el pasado (extrapolación retrógrada), en muchas áreas como la física, economía, biología, ingeniería (Isasi & Galván León, 2004).

Como ejemplo de una serie temporal, tomamos lo datos del consumo de energía eléctrica (kWh per cápita) en el Ecuador, mediante el cual se observa en la figura 1-2, la tendencia de la variable consumo de energía eléctrica en el período comprendido al año 1999 a 2016 (Galarza, 2017).

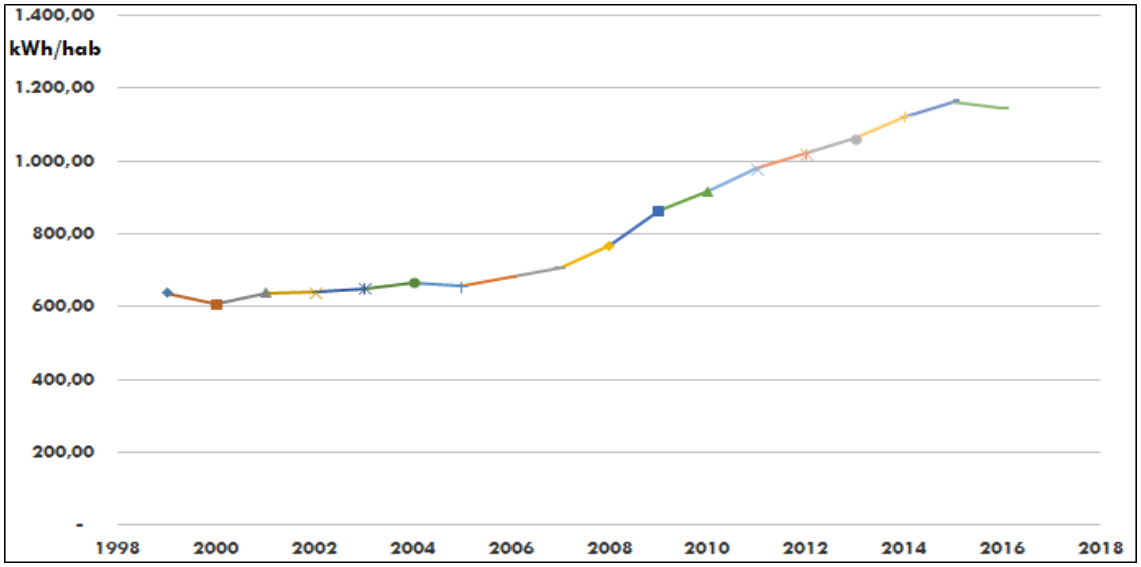


Figura 1-2: Consumo per cápita multianual (kWh/hab).
Fuente: ("Estadística anual y multianual del sector eléctrico", 2017)

Para describir el comportamiento de una variable generalmente se lo realizaba por medio de leyes físicas a través de la aplicación de técnicas clásicas y lineales, que expresan el valor de la serie en un instante de tiempo dado, como el caso de series que tengan comportamientos periódicos y estables, para otras series más complejas como comportamiento en el tiempo de fenómenos reales como la temperatura, radiación solar, lluvia, etc., dichas técnicas podrían resultar deficientes debido a la complejidad de establecer o generar el modelo que describa su comportamiento (Isasi & Galván León, 2004).

En este sentido las RNAs, han sido utilizadas para la predicción de series temporales ya que poseen las siguientes características:

- Capacidad de aproximación de relaciones a partir de un conjunto de ejemplo.
- Capacidad para construir relaciones no lineales.
- Capacidad para establecer relaciones con información incompleta y que incluso contenga ruido(Isasi & Galván León, 2004).

2.3. Neurona artificial

La unidad básica de un RNA es la *neurona*. Unos de los primeros modelos es la de tipo McCulloch-Pitts, propuesto por Warren McCulloch y Wlater Pitts en 1943. Mediante el cual modelaron el funcionamiento de las neuronas biológicas, es considerada un procesador elemental compuesta por X_n *entradas* y una *salida* S. Las entradas provienen del exterior o de neuronas a las que se encuentran conectadas. Poseen una función que les permite cambiar de nivel de activación de acuerdo con las señales que reciben, denominada *función de activación.* La entrada total a la neurona se calcula como la suma de todas las entradas ponderadas por ciertos valores. En la siguiente figura 2-2 puede verse la representación gráfica de la misma (Isasi & Galván León, 2004).

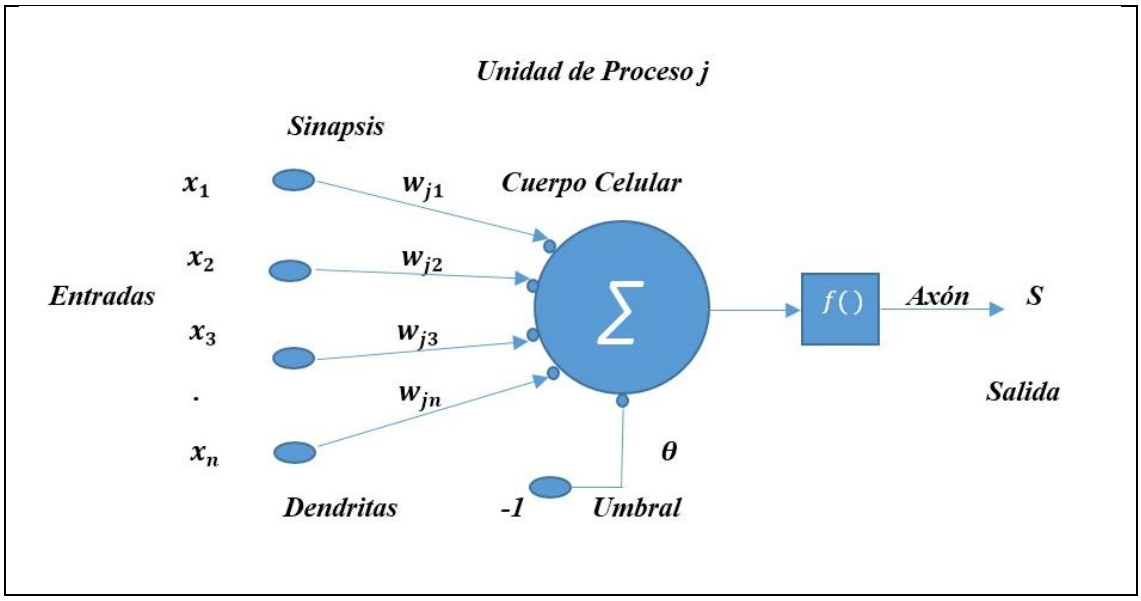


Figura 2-2: Esquema general de una célula Mcculloch-Pitts.
Fuente: (“Redes neuronales artificiales: un enfoque práctico”,2004)

Donde:

- $\boldsymbol{x_1}$**.....** $\boldsymbol{x_n}$, son las entradas.
- $\boldsymbol{w_{j1}}$**.....** $\boldsymbol{w_{jn}}$, son los pesos sinápticos.
- θ_j**,** umbral.
- $\boldsymbol{f()}$**,** función de activación.
- $\boldsymbol{S}$, salida.
- El *cuerpo celular*, o cuerpo central contiene el núcleo celular.
- EL *axón*, es una prolongación del cuerpo celular.
- Las *dendritas*, es una ramificación terminal.
- La *sinapsis*, es una conexión entre una neurona y otra

En la neurona de McCulloch-Pitts, el grupo de entradas $\boldsymbol{x_1 \ldots\ldots\ x_n}$, *(sinapsis de una neurona biológica)* son multiplicadas cada una por un peso al que está asociado $\boldsymbol{w_{j1} \ldots\ldots w_{jn}}$. El cuerpo de la neurona (Σ) suma todas las entradas algebraicamente y como resultado se obtiene una salida (E).

$$E = w_1 x_1 \ + \ w_2 \, x_2 + \ldots\ldots + w_n \qquad (1$$

La señal *(E)* se filtra a través de una función de activación, la cual genera una señal de salida de la neurona S. De acuerdo con la función de activación existirán distintos modelos de neuronas.

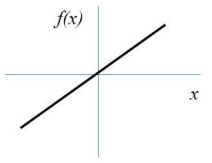
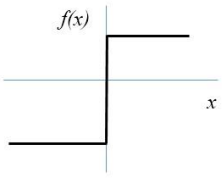
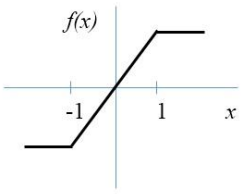
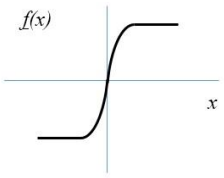
$$S = f(E - \theta) \qquad (2)$$

$$S = f(w_1 x_1 \ + \ w_2 \, x_2 + \ldots\ldots + w_n - \theta) \qquad (3)$$

Las funciones de activación más usuales son:

- Identidad
- Escalón
- Lineal a tramos
- Sigmoidea

Tabla 1-2: Funciones de activación.

	Función	Rango	Gráfica
Identidad	$y = x$	$[-\infty, +\infty]$	
Escalón	$y = sing\,(x)$ $y = H(x)$	$[-1, +1]$ $[0, +1]$	
Lineal a tramos	$-1, si\ x < -1$ $x, si\ +1 \leq x \leq -1$ $+1. si\ x > +1$	$[-1, +1]$	
Sigmoidea	$y = \frac{1}{1+e^x}$ $y = tgh\,(x)$	$[0, +1]$ $[-1, +1]$	

Fuente: ("Redes neuronales artificiales: un enfoque práctico", 2004)

2.4. Las redes neuronales

Las Redes Neuronales Artificiales (RNA) son modelos matemáticos y computacionales basados en el complejo comportamiento del cerebro humano, específicamente de las neuronas biológicas. La capacidad que tienen para aprender por medio de algoritmos de entrenamiento, han permitido su uso y aplicación en todas las ciencias (Isasi & Galván León, 2004).

La forma que se conecta una unidad neuronal con muchas otras se denomina arquitectura de red o conectividad, es decir es un grupo interconectado de neuronas análogo al cerebro humano. La RNA está conformada de varias capas, la más común es la red multicapa, conocida también como retropropagación, ya que el entrenamiento se realiza de adelante hacia atrás. La figura 3-2, muestra la estructura básica de una RNA, en la cual cada círculo representa una neurona artificial

las flechas muestran la conexión de la salida de una neurona a la entrada de otra. Las partes que conforman esta red se detallan a continuación (Isasi & Galván León, 2004).

Nivel 1: Capa de entrada, reciben los datos o señales externas (patrones de entrada).
Nivel 2: Capas ocultas, realizan el procesamiento interno de la red, puede haber una o varias.
Nivel 3: Capa de salida, genera la salida de la red.

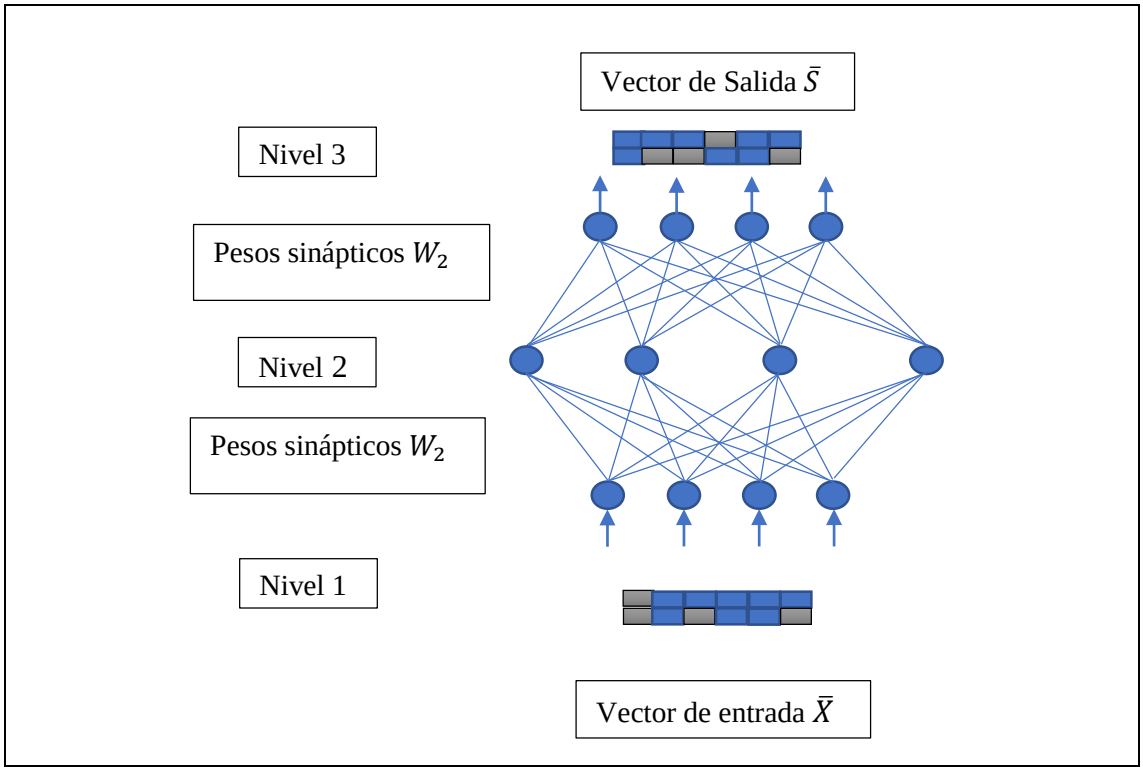


Figura 3-2: Esquema de una red de tres capas totalmente interconectadas.
Fuente: ("Redes neuronales artificiales: un enfoque práctico", 2004)

La información (valores numéricos) producida por cada neurona viaja a través de las conexiones son evaluados por los pesos respectivos, estos últimos se ajustan en la etapa de aprendizaje y como resultado de esto disponemos de una Red de Neuronas Artificial. El funcionamiento de la RNA presentada en la figura 3-3, se describe a continuación:

$$\bar{S} = F(F(\sum((\bar{X}\,.\,W_1) + \theta).W_2 \qquad (4)$$

Donde:

W_1 y W_2, son los pesos del primer y segundo nivel.

F, función de activación en todas las neuronas.

$\bar{X}$, vector de entrada a la red.

$\bar{S}$, vector de salida de la red.

2.5. El perceptrón multicapa

Fue desarrollado por Frank Rosenblatt en 1957, el cual añadió el aprendizaje al modelo de células de McCulloch-Pitts y lo denominó Percentrón. Está basado en la imitación del funcionamiento del ojo humano, una imagen proyectada por la retina se convierte en señales eléctricas por medio de las células sensitivas a la luz. Estas señales, a través de los axones de las células ganglionares se transmiten al cerebro e interpretan dicha información. En la figura 4-2, se presenta un modelo reducido de un perceptrón (Isasi & Galván León, 2004).

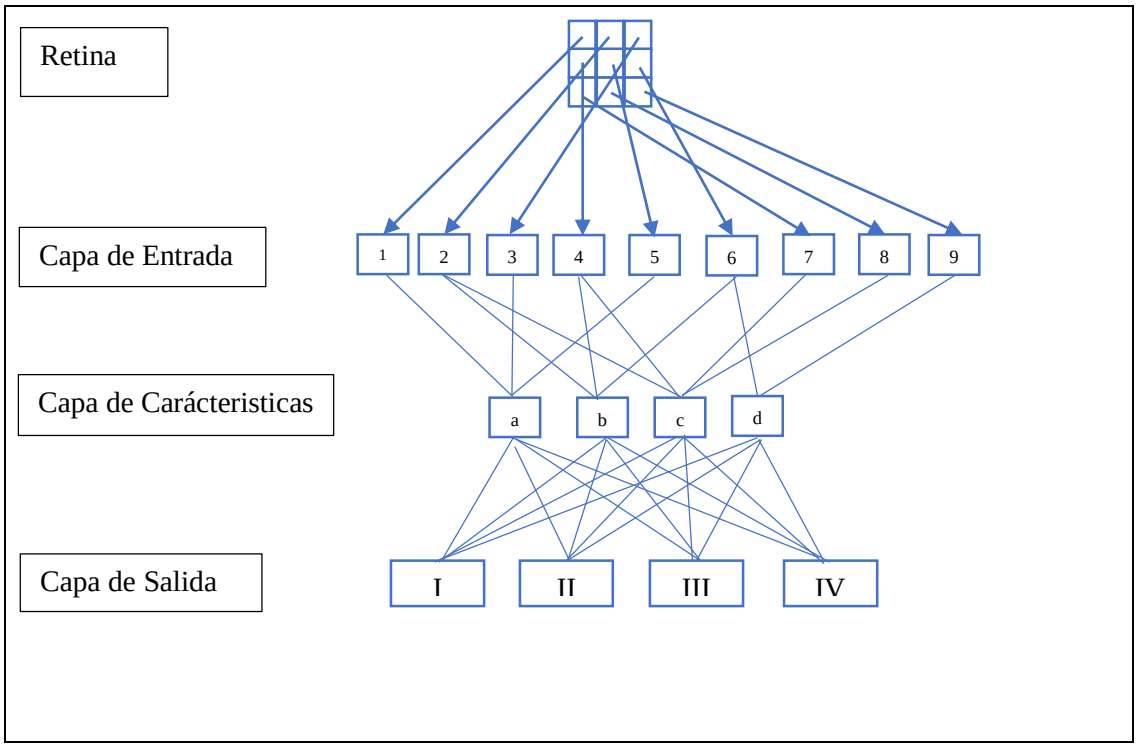


Figura 4-2: Modelo del perceptrón.
Fuente: ("Redes neuronales artificiales: un enfoque práctico", 2004)

Como se observa en la figura 5-2, el perceptrón multicapa está constituido por un mínimo de tres capas que se detallan a continuación:

- Capa de entrada, reciben las señales o patrones del exterior y las propagan a la siguiente capa.
- Capa oculta, pueden ser una o varias capas, se encargan del procesamiento no lineal de patrones.
- Capa de salida, remite al exterior la salida de la red de cada patrón recibido(Isasi & Galván León, 2004).

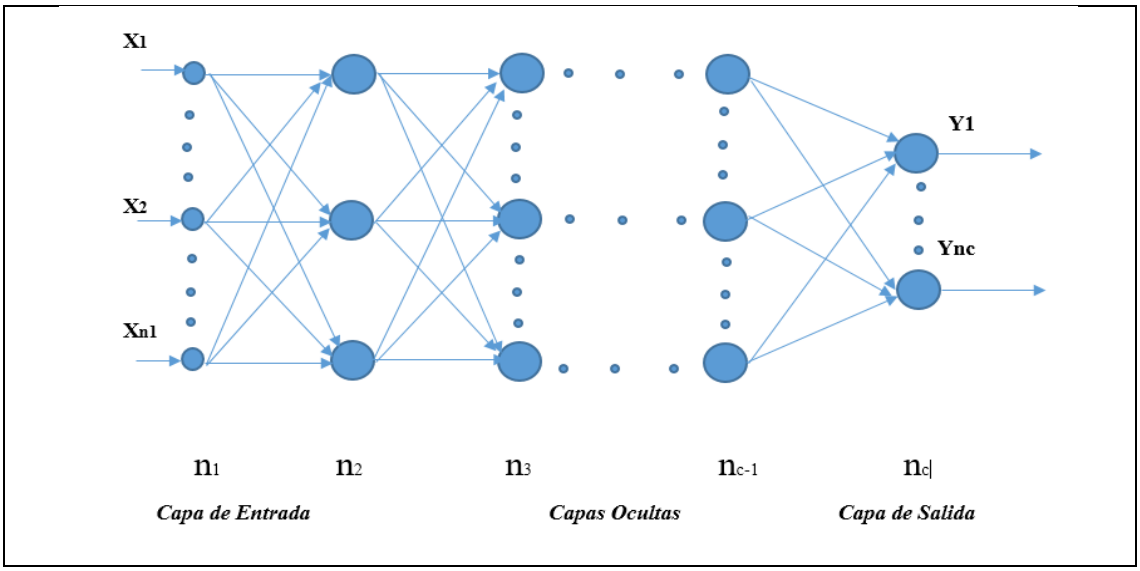


Figura 5-2: Modelo del perceptrón.
Fuente: ("Redes neuronales artificiales: un enfoque práctico", 2004)

Por su manera de flujo de información toman el nombre de redes feedforward (hacia delante), las conexiones tienen asociado un peso y todas las neuronas tienen un umbral, para el caso del perceptrón es una conexión más de la neurona con un valor (umbral) de 1. Una de sus principales características es que se encuentra totalmente conectada es decir las neuronas de la capa de entrada están conectadas a las neuronas de la primera capa oculta, las neuronas de la capa oculta están conectadas a las neuronas de la siguiente capa, etc. Generalmente este tipo de estructura utiliza las funciones de activación sigmoidea y tangente hiperbólica descritas en la tabla 1-2 (Isasi & Galván León, 2004).

2.6. Red neuronal recurrente

Este tipo de redes se caracterizan por crear bucles en las neuronas de la red mediante el uso de conexiones recurrentes, a diferencia de la red perceptrón. Este tipo de conexiones (bucles) se muestra en la figura 6-2 (Isasi & Galván León, 2004), y se detallan a continuación:

a. Conexión de neurona en sí misma.
b. Conexiones entre neuronas de una misma capa.
c. Conexiones de las neuronas de una capa a la capa anterior.

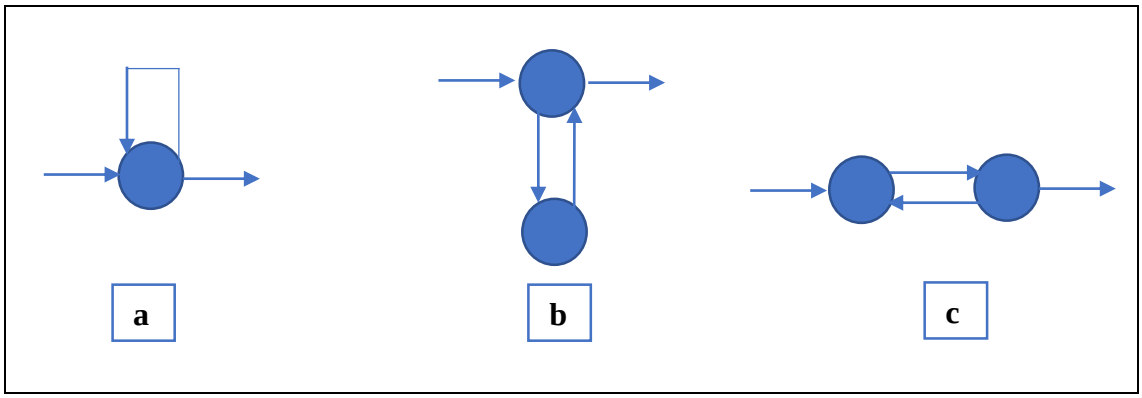


Figura 6-2: Ejemplos de neuronas con conexiones recurrentes.
Fuente: (“ Redes neuronales artificiales: un enfoque práctico”, 2004)

Al incluir conexiones recurrentes (bucles), la activación de una neurona ya no solo depende de las activaciones de las neuronas de la capa anterior (como en el perceptrón multicapa), sino también del estado de activación de cualquier otra neurona de la red a la que se encuentra conectada. En este sentido es importante incluir la variable tiempo en la activación o estado de la neurona, la cual está dada por (Isasi & Galván León, 2004).

$$a_i(t+1) = f_i(\sum_j w_{ji} a_j(t)) \qquad (5)$$

Donde:
El índice ***j,*** varía en el conjunto de todas las neuronas conectadas a la neurona ***i***. La presencia de la variable *tiempo* ***(t)*** en las activaciones de las neuronas recurrentes, hace que estas redes posean un comportamiento dinámico o temporal. Por tanto, este tipo de redes son una herramienta utilizada para modelar series temporales (Isasi & Galván León, 2004).

Existen tres tipos de redes recurrentes y se detallan a continuación.

2.6.1. Red Hopfield

Desarrollada por Jhon Hopfield, en 1980, se presenta como un modelo de memoria asociativa de patrones o muestras, ya que es capaz de recuperar patrones almacenados a partir de información incompleta sobre los patrones inconclusos a partir de patrones con ruido. Su característica indica que todas las neuronas están conectadas con todas las demás salvo a ella mismo. Como se observa en la figura 7-2 (Isasi & Galván León, 2004).

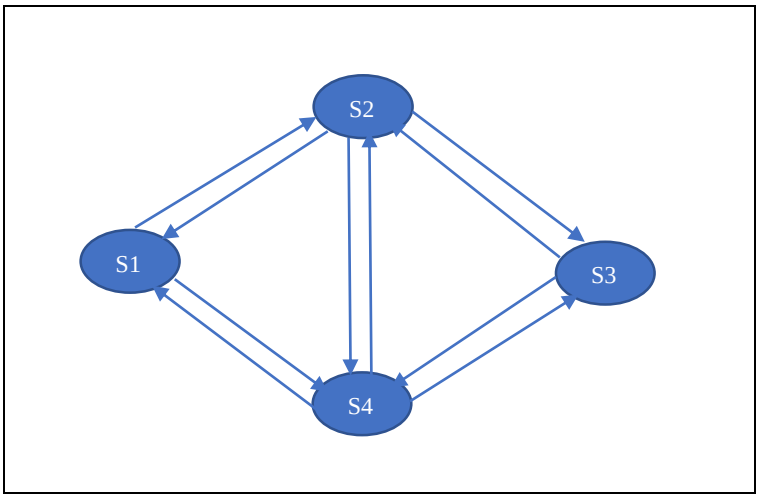


Figura 7-2: Esquema de la Red Hopfield.
Fuente: ("Redes neuronales artificiales un enfoque práctico", 2004)

2.6.2. Red parcialmente recurrente

Son redes multicapa y se caracterizan por que tiene dos tipos de entradas de neuronas a la red, la primera son las neuronas de entrada que se encargan de recibir la información del exterior y la segunda son las neuronas de contexto que se encargan de recoger información de conexiones recurrentes y funcionan como memoria de la red como se observa en la figura 8-2. Las conexiones recurrentes en las redes parcialmente recurrentes suelen ser conexiones uno a uno, es decir, de una neurona de la red a una única neurona de contexto (Isasi & Galván León, 2004).

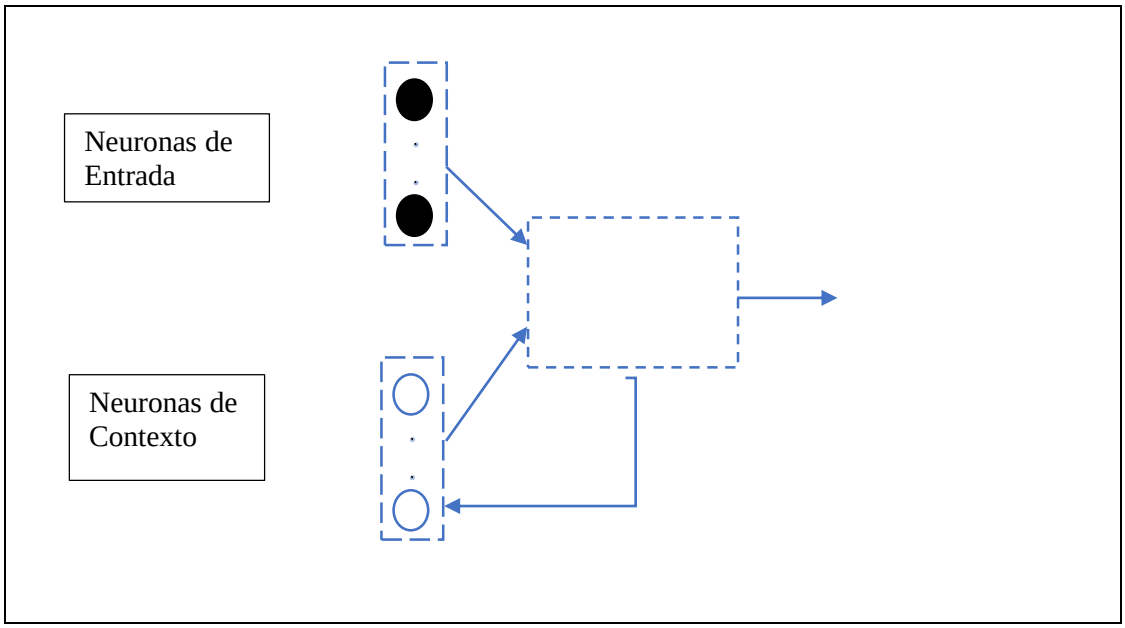


Figura 8-2: Esquema de una red parcialmente recurrente.
Fuente: ("Redes neuronales artificiales un enfoque práctico", 2004)

2.6.3. Red Totalmente Recurrente.

Las redes totalmente recurrentes se caracterizan debido a que las neuronas de su red reciben como entradas la activación del resto de las neuronas de la red, así como su propia activación. El esquema de esta red se ilustra en la siguiente figura 9-2 (Isasi & Galván León, 2004).

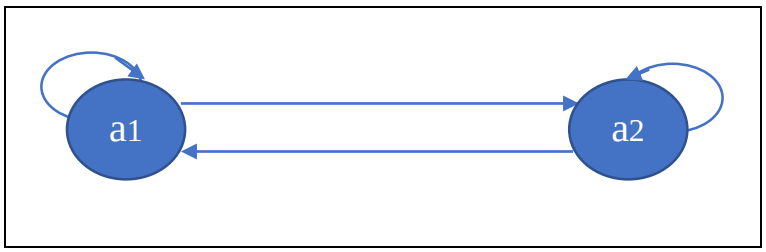


Figura 9-2: Esquema de una red totalmente recurrente.
Fuente: ("Redes neuronales artificiales: un enfoque práctico", 2004)

2.7. Librería Keras

Desarrollada por François Chollet, Keras es una librería de redes neuronales escrita en Python y capaz de ejecutarse tanto sobre MXNet, Deeplearning4j, Tensorflow o Theano. Sus principales características son: (Chollet, 2015b).

- Soporta tanto redes convolucionales como recurrentes y la combinación de ambas.
- Soporta esquemas de conectividad arbitraria (incluyendo entrenamiento multientrada y multisalida).
- Puede ser ejecutada con los recursos del CPU y GPU
- Es compatible con Python 2.7 al 3.5.

2.7.1. Modelos

Existen dos tipos de modelos disponibles para implementar en Keras: el modelo secuencial, y la clase Model (más general). Ambos modelos presentan varios métodos en común, como summary (), get_config() o to_json() que devuelven información básica sobre el modelo, o get_weights() y set_weights(), los métodos getter() y setter() propios de los pesos del modelo (Cortés, 2017).

2.7.2. Capas

Para diseñar las capas de una red neuronal, se dispone de las siguientes funciones:
Dense: Para capas regulares totalmente conectadas

Activation: Se aplica una función de activación a la salida.

2.7.3. Funciones de activación

- *softmax*: generalización de la función logística.
- *relu:* función rectificadora.
- *tanh:* función tangente hiperbólica.
- *sigmoid:* función sigmoide.
- *linear*: función lineal.

2.7.4. Inicializadores

Definen la manera de inicializar los pesos de las capas de la red. El argumento usado para definir este inicializador en las capas es init. Las opciones más comunes son: (Chollet, 2015a).

- uniform
- normal
- identity
- zero
- glorot_normal
- glorot_uniform

2.7.5. Funciones de pérdida

La función de pérdida es uno de los dos parámetros necesarios para compilar un modelo (Chollet, 2015b). Las más conocidas que proporciona Keras es:

mean_squared_error(y_true, y_pred) : Calcula el error cuadrático medio.
mean_absolute_error/mae: Calcula el error medio absoluto.
binary_crossentropy: Calcula la pérdida logarítmica.

2.7.6. Algoritmos de entrenamiento

El estudio del arte permite identificar en la herramienta Keras 7 algoritmos que se pueden utilizar para el análisis de algoritmos de entrenamiento de redes neuronales artificiales, aplicadas al

modelamiento dinámico de represas hidroeléctricas, mediante el error de predicción del nivel de embalse de agua, y que se detallan a continuación.

2.7.6.1. *Descenso de Gradiente Estocástico SGD.*

Este algoritmo es simple y bien conocido, en general es un algoritmo de optimización muy robusto, en el cual el gradiente de la función que se minimiza con respecto a los parámetros calculados $\nabla f(\theta_{t-1})$, una parte de η del gradiente es restados fuera de los parámetros (Dozat, 2016).

Algoritmo: Descenso del Gradiente

$$g_t \leftarrow \nabla_{\theta_{t-1}} f(\theta_{t-1}) \quad (6)$$

$$\theta_t \leftarrow \theta_{t-1} - \eta g_t \quad (7)$$

Momento clásico, acumula una suma de decaimientos (con la constante de decaimiento μ) de los gradientes anteriores en un vector **m.** Esto ayuda a acelerar el aprendizaje del descenso de gradiente.

Algoritmo: Momentun clásico

$$g_t \leftarrow \nabla_{\theta_{t-1}} f(\theta_{t-1}) \quad (8)$$

$$\boldsymbol{m}_t \leftarrow \mu \boldsymbol{m} \theta_{t-1} + g_t \quad (9)$$

$$\theta_t \leftarrow \theta_{t-1} - \eta m_t \quad (10)$$

Momento Nesterov, este algoritmo probablemente tiene un mejor desempeño que el gradiente descendente, podría ser descrito como una clase de momento mejorado.

$$\theta_t \leftarrow \theta_{t-1} - \eta m_t \quad (11)$$

$$\theta_t \leftarrow \theta_{t-1} - \eta m_{t-1} - \eta g_t \quad (12)$$

Observamos que el término m_{t-1}, no depende de la pendiente del gradiente g_t. En principio podemos obtener un paso en la dirección superior aplicando el vector de momento a los parámetros antes de calcular el gradiente.

Algoritmo: Gradiente Acelerado Nesterov.

$$g_t \leftarrow \nabla_{\theta_{t-1}} f(\theta_{t-1} - \eta\mu m_t) \quad (13)$$

$$\boldsymbol{m}_t \leftarrow \mu\boldsymbol{m}_{t-1} + g_t \quad (14)$$

$$\theta_t \leftarrow \theta_{t-1} - \eta m_t \quad (15)$$

2.7.6.2. AdaGrad

El algoritmo adaptativo del descenso del gradiente divide el valor de η en cada paso de la norma L_2 , de todos los gradientes anteriores. Esto permite que el aprendizaje se reduzca en dimensiones que han cambiado significativamente y acelere el aprendizaje en las dimensiones que han cambiado ligeramente.

Algoritmo: Adagrad

$$g_t \leftarrow \nabla_{\theta_{t-1}} f(\theta_{t-1}) \quad (16)$$

$$\boldsymbol{n}_t \leftarrow \boldsymbol{n}_{t-1} + g_t^2 \quad (17)$$

$$\theta_t \leftarrow \theta_{t-1} - \eta \frac{g_t}{\sqrt{\eta_t + \varepsilon}} \quad (18)$$

Un notable problema con este algoritmo es que la norma del vector **n** llega a ser muy grande que el entrenamiento se desacelera hasta detenerse, sin encontrar el mínimo local.

2.7.6.3. Adadelta

Es una extensión del algoritmo Adagrad, que busca reducir su agresividad, disminuyendo monótonamente la tasa de aprendizaje (Ruder, 2016). La función se determina a continuación (Zeiler, 2012).

Algoritmo: Adadelta

Requiere: Taza de aprendizaje global $\in$ (valor sugerido: 0.001)

Requiere: Parámetro inicial pesos θ

Inicializa la acumulación de variables $E[g^2]_0 = 0 \;\; E[\Delta\theta^2]_0 = 0$ *(19)*

for t = 1: T do %% laso hasta el # de iteraciones

Calcula la estimación del gradiente: $\hat{g} \leftarrow \frac{1}{m}\nabla_\theta \sum_i L(f(x^i;\theta),), y^{(i)})$ *(20)*

Acumula el gradiente: $E[g^2]_t = \rho E[g^2]_{t-1} + (1-\rho)g_t^2$ *(21)*

Calcula la actualización: $\Delta x_t = -\frac{RMS\,[\Delta\theta]_{t-1}}{RMS[g]_t}\, g_t$ *(22)*

Acumulación de la actualización: $E[\Delta\theta^2]_t = \rho E[\Delta\theta^2]_{t-1} + (1-\rho)\Delta\theta_t^2$ *(23)*

Aplica la actualización de los pesos: $\theta_{t+1} = \theta_t + \Delta\theta_t$ *(24)*

End While

2.7.6.4. RMSprop

Este algoritmo de aprendizaje es una alternativa del algoritmo Adagrad, que remplaza la suma en $\boldsymbol{n}_t$ con una disminución del parámetro ***v.*** Esto permite que el modelo continúe aprendiendo indefinidamente.

Algoritmo: RMSprop

$g_t \leftarrow \nabla_{\theta_{t-1}} f(\theta_{t-1})$ *(25)*

$\boldsymbol{n}_t \leftarrow \boldsymbol{v}\boldsymbol{n}_{t-1} + (1-v)g_t^2$ *(26)*

$\theta_t \leftarrow \theta_{t-1} - \eta\frac{g_t}{\sqrt{\eta_t + \varepsilon}}$ *(27)*

2.7.6.5. Adam

El análisis de la combinación de los métodos basados en momento y basadas en norma ha demostrado tender un buen desempeño. Como es el caso del algoritmo Adam que combina el momento clásico con el algoritmo RMSprop para obtener un mejor rendimiento. Su algoritmo incluye términos de corrección del sesgo inicial. Se compensa la inestabilidad inicializando los parámetro **m** y **n** con 0 (Dozat, 2016).

Algoritmo 4: Adam

$\widehat{n_t} \leftarrow \frac{n_t}{1 - v^t}$ *(28)*

$$m_t \leftarrow \boldsymbol{u}\boldsymbol{m}_{t-1} + (1-u)g_t \quad (29)$$

$$\widehat{m_t} \leftarrow \frac{m_t}{1-u^t} \quad (30)$$

$$\boldsymbol{n}_t \leftarrow \boldsymbol{v}\boldsymbol{n}_{t-1} + (1-v)g_t^2 \quad (31)$$

$$g_t \leftarrow \nabla_{\theta_{t-1}} f(\theta_{t-1}) \quad (32)$$

$$\theta_t \leftarrow \theta_{t-1} - \eta \frac{\widehat{m_t}}{\sqrt{\eta_t + \varepsilon}} \quad (33)$$

2.7.6.6. *Adamax*

Es una variante del algoritmo Adam basada en la norma del infinito. Reemplaza la norma L_2 con la norma L_∞, eliminando $\widehat{n_t}$ y sustituyéndolo por n_t y θ_t con los siguientes cambios:

$$n_t \leftarrow \max(vn_{t-1}, |g_t|) \quad (34)$$

$$\theta_t \leftarrow \theta_{t-1} - \eta \frac{\widehat{m_t}}{\eta_t + \varepsilon} \quad (35)$$

Algortimo: Adamax

$$g_t \leftarrow \nabla_{\theta_{t-1}} f(\theta_{t-1}) \quad (36)$$

$$m_t \leftarrow \boldsymbol{u}\boldsymbol{m}_{t-1} + (1-u)g_t \quad (37)$$

$$\widehat{m_t} \leftarrow \frac{m_t}{1-u^t} \quad (38)$$

$$n_t \leftarrow \max(vn_{t-1}, |g_t|) \quad (39)$$

$$\widehat{n_t} \leftarrow \frac{n_t}{1-v^t} \quad (40)$$

$$\theta_t \leftarrow \theta_{t-1} - \eta \frac{\widehat{m_t}}{\eta_t + \varepsilon} \quad (41)$$

2.7.6.7. *Nadam*

Este algoritmo de aprendizaje es el resultado de la combinación del algoritmo NAG con el Adam. La función se detalla a continuación:

Algoritmo: Nadam

$$g_t \leftarrow \nabla_{\theta_{t-1}} f(\theta_{t-1}) \quad (42)$$

$$\hat{g} \leftarrow \frac{g_t}{1 - \prod_{i=1}^{t+1} u_i} \quad (43)$$

$$m_t \leftarrow \boldsymbol{u}\boldsymbol{m}_{t-1} + (1-u) g_t \quad (44)$$

$$\widehat{m_t} \leftarrow \frac{m_t}{1 - \prod_{i=1}^{t+1} u_i} \quad (45)$$

$$\boldsymbol{n}_t \leftarrow \boldsymbol{v}\boldsymbol{n}_{t-1} + (1-v) g_t^2 \quad (46)$$

$$\widehat{n_t} \leftarrow \frac{n_t}{1 - v^t} \quad (47)$$

$$\overline{m_t} \leftarrow (1 - u_t)\widehat{g_t} + u_t + 1\widehat{m_t} \quad (48)$$

$$\theta_t \leftarrow \theta_{t-1} - \eta \frac{\overline{m_t}}{\sqrt{\widehat{n}_t} + \varepsilon} \quad (49)$$

2.8. Redes utilizadas como modelo

El modelamiento de un proceso dinámico es la capacidad para crear un esquema o modelo que permita simular de forma matemática y computacional el comportamiento de dicho proceso. Algunos procesos dinámicos se pueden obtener con la ayuda de las leyes de la física que gobiernan dicho modelo. Sin embargo, en ocasiones resulta difícil modelar un proceso dinámico debido que a se debe considerar todas las leyes físicas que intervienen en un proceso (Isasi & Galván León, 2004).

El control de un proceso dinámico consiste en regular el comportamiento de un proceso por medio de la manipulación de las variables de entrada al proceso, de esta manera un sistema de control se encarga de proporcionar la acción que hay que aplicar al proceso para que la salida de dicho proceso tienda al valor deseado (Isasi & Galván León, 2004).

Las RNAs permiten desarrollar técnicas de control para procesos dinámicos no lineales, es decir la red es la encargada de calcular la acción de control de un proceso. Los distintos métodos de control dependen de la forma de realizar el entrenamiento de la red, así como el tipo de red de neuronas a utilizar.

A continuación, se detallan los diferentes esquemas de sistemas de control basados en redes neuronales:

2.8.1. Copia de un controlador ya existente

La red aprende el comportamiento de un controlador, utilizando como salida deseada para la red, la salida de dicho controlador, es decir copia un controlador ya existente.

2.8.2. Control Inverso

Consiste en aproximar mediante una red de neuronas la dinámica inversa de un proceso. Se lo lleva a cabo mediante dos formas diferentes:

2.8.2.1. *Aprendizaje generalizado.*

Se utiliza un conjunto de datos representativo de una dinámica inversa, los cuales se obtienen manipulando la acción de control en sus diferentes etapas de operación y midiendo la salida.

En la figura 10-2, se muestra el esquema general de aprendizaje. La entrada al proceso actúa como salida deseada para la red, mientras la salida del proceso es la entrada para la red. El aprendizaje de la red se realiza siguiendo la dirección negativa del gradiente de la siguiente función de error:

$$e_g(t) = \frac{1}{2}(\tilde{u}(t) - u(t))^2 \qquad (50)$$

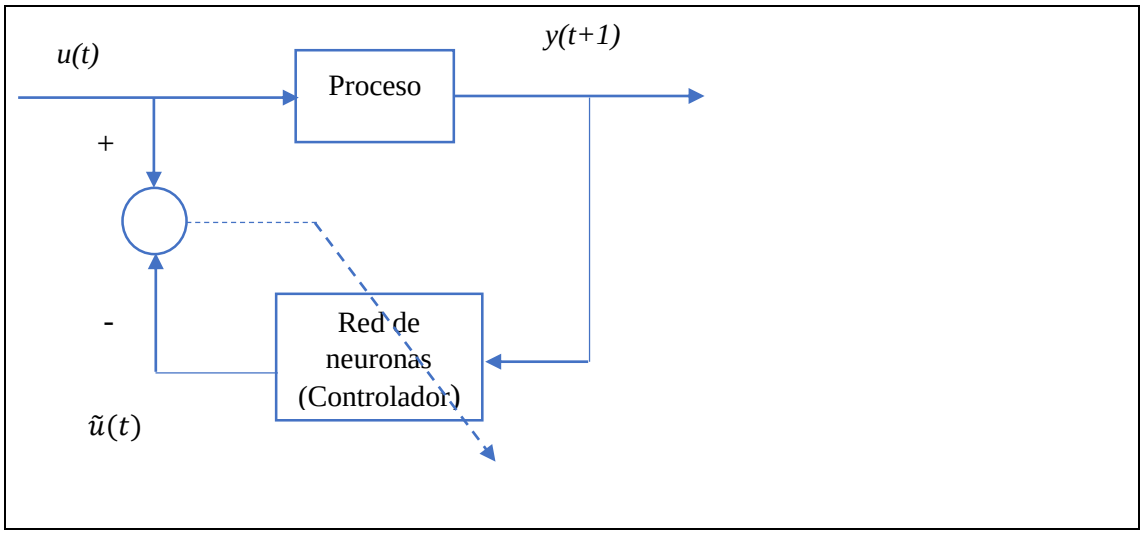


Figura 10-2: Esquema control inverso con aprendizaje generalizado.
Fuente: ("Redes neuronales artificiales: un enfoque práctico", 2004)

2.8.2.2. Aprendizaje especializado.

La red aproxima la dinámica inversa local del proceso, es decir, la dinámica inversa en una determinada región de interés y no en todas las etapas de operación de control. En la siguiente figura 11-2 la entrada a la red es el objetivo de control y la salida de la red es la acción de control que se aplica al proceso dinámico (Isasi & Galván León, 2004).

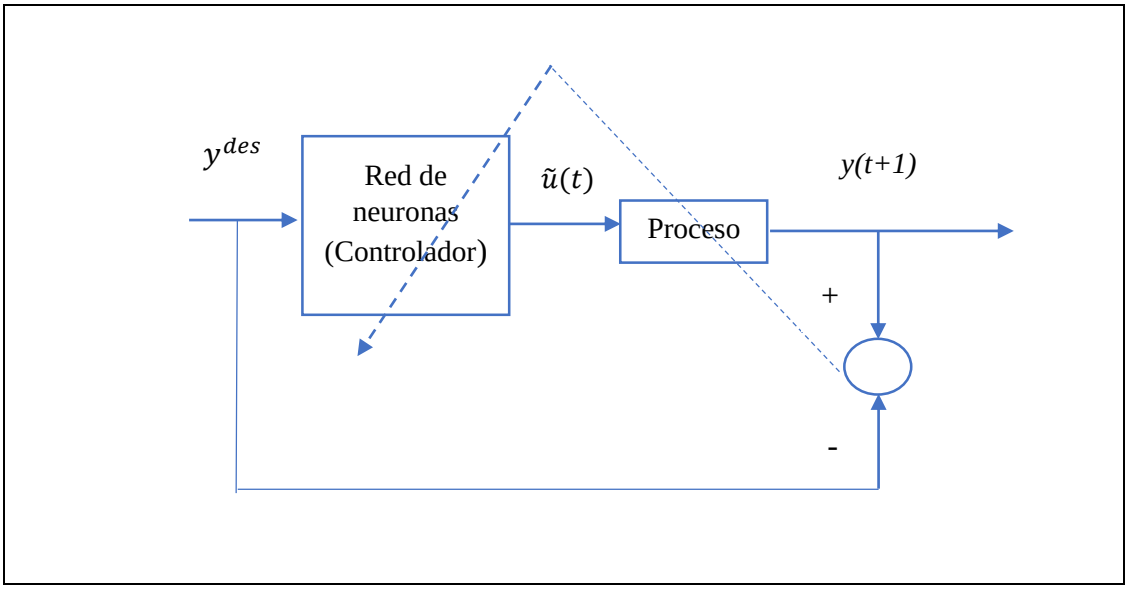


Figura 11-2: Esquema de control inverso con aprendizaje especializado.
Fuente: ("Redes neuronales artificiales: un enfoque práctico", 2004)

El aprendizaje de la red se realiza para minimizar la diferencia entre la salida del proceso y el objetivo de control deseado, es decir:

$$e_s(t+1) = \frac{1}{2}(y^{des} - y(t+1))^2 \qquad (51)$$

2.8.3. Control Predictivo

Esta estrategia de control utiliza el comportamiento dinámico del proceso en el futuro para determinar la acción de control en el instante actual. Para predecir el comportamiento futuro y calcular la acción de control actual se utiliza un modelo de proceso. De acuerdo a la figura 12-2, es similar al control inverso con aprendizaje especializado, ya que la red de neuronas tiene como entrada el objeto de control y su salida es la acción de control que hay que aplicar al proceso dinámico (Isasi & Galván León, 2004). El aprendizaje de la red se realiza para minimizar las diferencias entre el objetivo de control y la salida de un modelo de proceso en el futuro, es decir:

$$e_p(t+1) = \frac{1}{2}\sum_{h=0}^{H}(y^{des} - \tilde{y}(t+h+1))^2 \qquad (52)$$

La acción de control no solo se calcula a partir de la respuesta del proceso en el instante de tiempo, sino también a partir del proceso en un futuro más lejano, dado por el horizonte de predicción *h*.

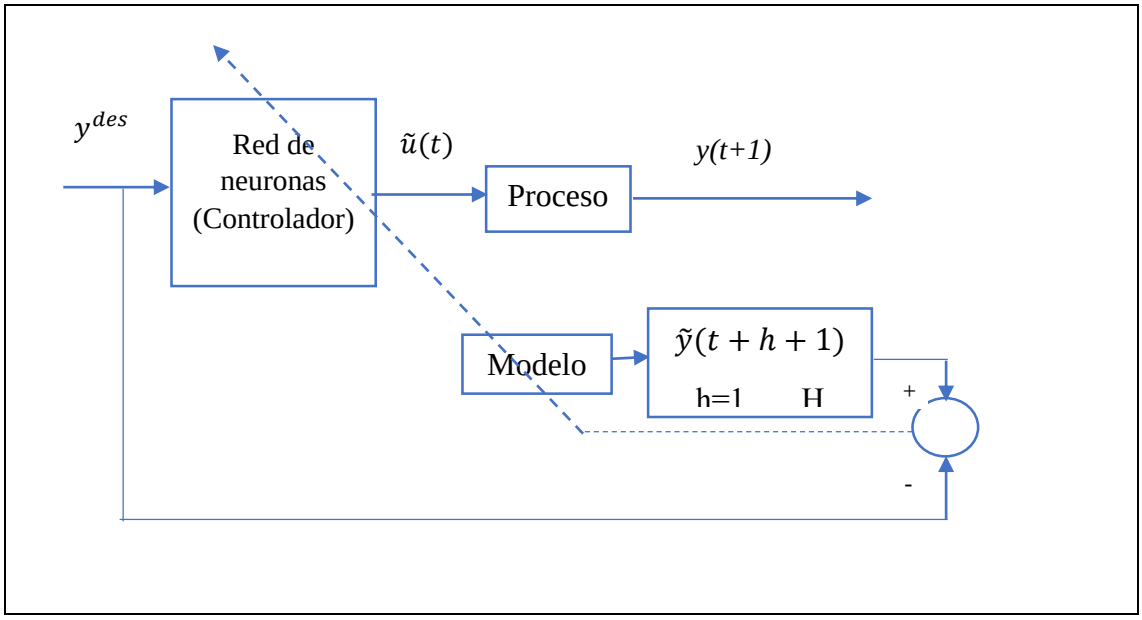


Figura 12-2: Esquema de control predictivo.
Fuente: ("Redes neuronales artificiales: un enfoque práctico", 2004)

CAPÍTULO III

3. METODOLOGÍA DE LA INVESTIGACIÓN

El propósito de este capítulo es detallar mediante fases, la metodología utilizada en el entrenamiento de RNAs, por medio del uso y aplicación de algoritmos de entrenamiento proporcionados por la librería Keras de Python, con el fin de determinar cuál de estos tiene un mejor desempeño en base al error de entrenamiento, validación y umbral de predicción. Además, se medirá el tiempo que toma el entrenamiento de dichos algoritmos.

En este caso de estudio, la información utilizada proviene de la Central Hidroeléctrica Agoyán, ubicada en la provincia del Tungurahua a 180 Km. Al sureste de la ciudad de Quito y a 5 km al este de la ciudad de Baños en el sector denominado Agoyán de la parroquia Ulba, concebida para aprovechar el caudal de río Pastaza, cuenta con una producción media anual de 1.010 (GWh/año), una potencia efectiva y nominal de 156 MW y 160 MW respectivamente y un factor de planta de 73,90% (CONELEC, 2013).

Para analizar el desempeño de los algoritmos de entrenamiento de RNAs, de la librería Keras, fue necesario desarrollar un aplicativo de software que permita registrar, visualizar, tanto el error de entrenamiento, error de validación y el error de predicción de nivel de embalse de agua, esta implementación se realizó de acuerdo con las fases que se detallan en la figura 1-3.

Figura 1-3: Fases de la implementación.
Realizado por: Edison Chafla, 2019.

La figura 2-3 muestra la secuencia de procedimientos desarrollados para implementación del aplicativo de software.

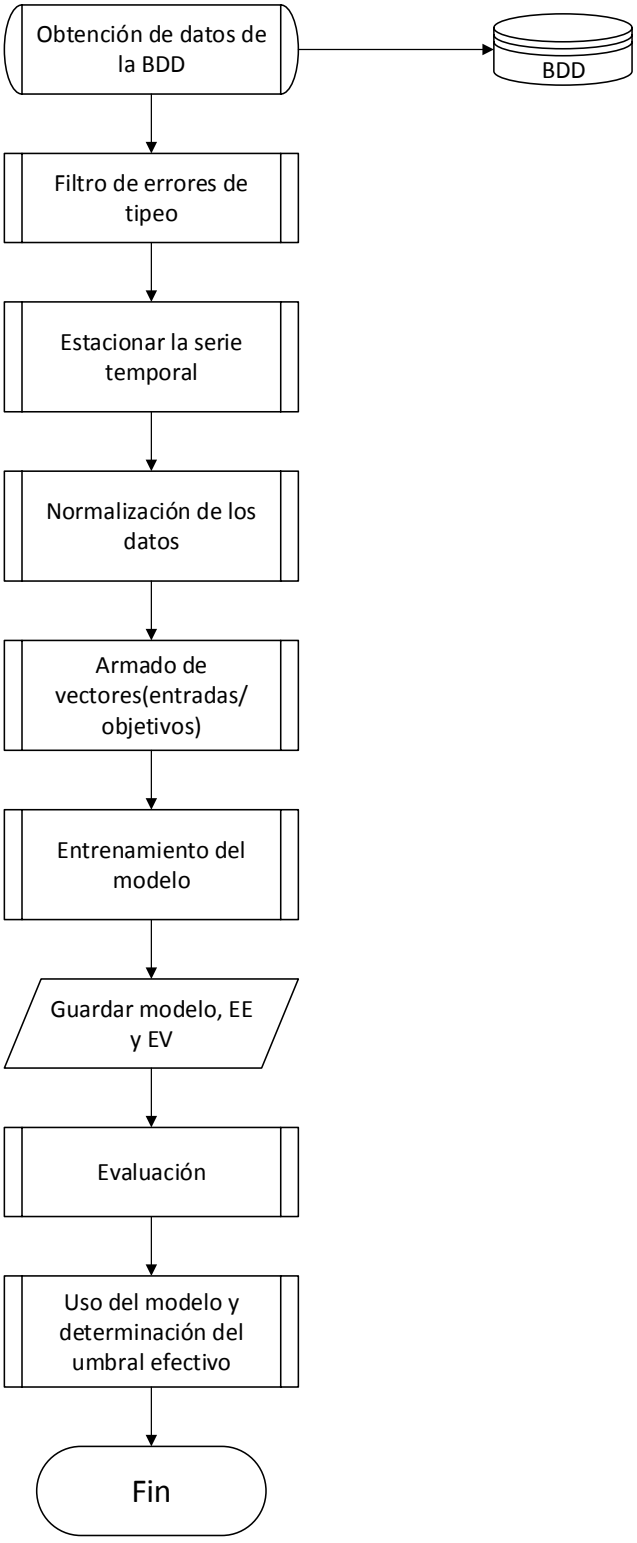


Figura 2-3: Diagrama de flujo de la implementación del aplicativo de software.
Realizado por: Edison Chafla, 2019.

3.1. Preparación y análisis de datos

La información utilizada corresponde a los datos de nivel, caudal y potencia activa total generada, de la Hidroeléctrica Agoyán, del período comprendido entre el 1/01/2005 hasta el 31/04/2016. La mismas fueron adquiridas mediante un sistema SCADA (Supervisory Control And Data Acquisition), que obtiene los datos de los sensores de nivel de embalse y de caudal del río Pastaza, a la cual se realiza la validación y preprocesamiento de los datos antes de utilizarlos en los algoritmos de aprendizaje (Asqui, 2017).

3.1.1. Base de Datos

La data se encuentra almacenada en la base de datos desarrollada en el proyecto de investigación de G. Asqui, y alojada en el gestor de base de datos PostgreSQL, se realizó un único cambio en el identificador o primary key, debido a que, en la anterior base de datos el identificador fue un número bigint (número de 8 bytes), actualmente se utiliza a la fecha como identificador ya que cada dato tiene una única fecha y hora, por tal motivo se eliminó la columna del bigint. Lo que implica menores tiempos en recuperación de datos y menor tamaño en la base de datos.

El método de acceso a los datos desde el aplicativo se realiza mediante el uso de la librería Psycopg, que enlaza Python con PostgreSQL, se encuentra detallado en el Anexo 1.

3.1.2. Datos totales

La base de datos está compuesta por un total de 102.702 datos por cada variable (ver Tabla 3-1), estas fueron recogidas por los operadores de la central hidroeléctrica Agoyán, desde el 01-01-2005 hasta el 31-04-2016, se detalladas en la tabla 1-3.

Tabla 1-3: Variables

Variable	**Descripción**	**Unidad de medida**
Nivel de embalse	Metros sobre el nivel del mar	m.s.n.m.
Caudal	Metros cúbicos por segundo	m^3/s
Potencia total activa	Potencia total activa genera en Megavatios	Mw

Fuente: Hidroeléctrica Agoyán, 2018
Realizado por: Edison Chafla, 2019.

3.1.3. Temporadas climáticas

El análisis efectuado a la variable caudal, dentro de la empresa, identificó que esta tiende a repetirse cada año, por tal motivo se asignó el término de "Temporadas Climáticas", definiendo 3 tipos de estas, para el desarrollo de la plataforma se estableció una numeración a cada temporada como se presenta en la tabla 2-3.

Tabla 2-3: Temporadas Climáticas

Período	Temporada	Meses
Invernal Fuerte	1	Marzo – Julio
Invernal Ligera	2	Agosto – Octubre
Verano	3	Noviembre - Febrero

Fuente: Hidroeléctrica Agoyán
Realizado por: Edison Chafla, 2019.

3.1.4. Filtro de errores de tipeo

La primera validación realizada a los datos contempló la eliminación de rangos de cada variable, con el objeto que la RNA a entrenar no registre estos parámetros y entregue comportamientos ilógicos, de acuerdo al siguiente detalle:

- Se eliminan datos de caudal $< 35 \frac{m^3}{s}$.
- Se eliminan datos de potencia activa $> 156MW$.
- Se eliminan datos de nivel $> 1651\ m.s.n.m.$

3.1.5. Estacionar la serie temporal

El análisis realizado a la variable del caudal determinó que es una serie temporal no estacionaria, para conseguir que esta variable se aproxime a una serie temporal estacionaria, fue necesaria la aplicación de la función logarítmica a la señal de caudal, también se analizó como esta transformación ayudó a acentuar los picos del PDS (Densidad espectral de potencia) de la señal de caudal (Asqui, 2017).

3.1.6. Armado de ventanas, basado en PSD

Para poder determinar el error de predicción de nivel de agua de embalse N+1 como se presenta en la figura 3.3, fue necesario establecer de manera aproximada el número de neuronas de entrada y el número de capas ocultas de la RNA, y posterior aplicar los algoritmos de entrenamiento de

la librería Keras de Python. Esto se consigue mediante el análisis de frecuencia de las 3 señales de entrada a la RNA, las cuales son nivel, caudal y potencia a producir (Zaldivar, Gutiérrez, & Galván, 2000).

El método mediante el cual se realiza el análisis de frecuencia es por medio de la densidad espectral de potencia (PSD, siglas en inglés, power density spectral), muestra a que frecuencias las variaciones son más fuertes y en que frecuencias las variaciones son más débiles, de esta manera se combina el número de neuronas de acuerdo a los picos del PSD, para las entradas como para las capas ocultas (J. Correa, Morales, Huerta, González, & Cárdenas, 2016).

La configuración utilizada para el modelo de la presa fue un esquema MISO (Multiple input single output), cuando el sistema tiene varias entradas y salida única, en este caso las entradas son las variables caudal, nivel y potencia, como salida obtenemos el nivel futuro en una hora, como se identifica en la figura 3-3 (Azagra, 2017).

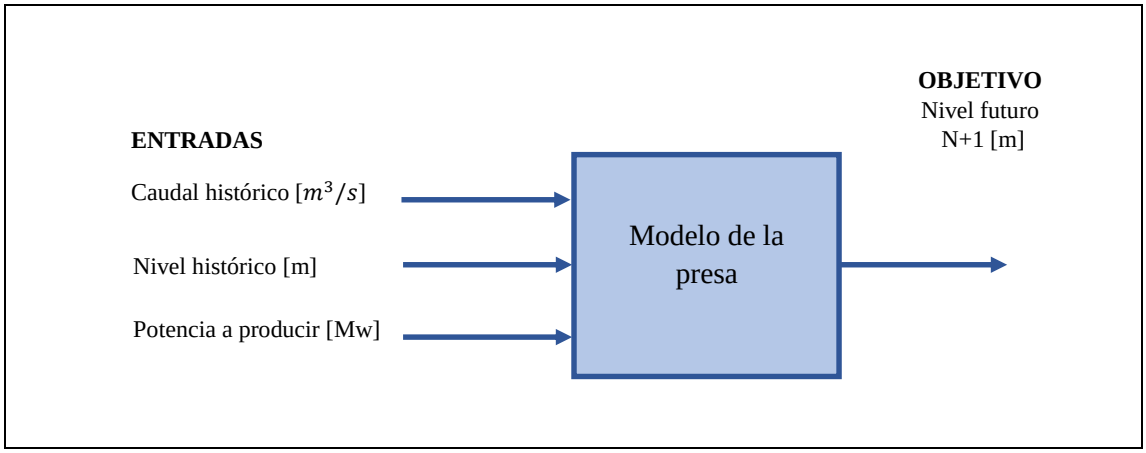


Figura 3-3: Diagrama del modelo de la presa.
Realizado por: Edison Chafla, 2019.

Del análisis realizado se ha determinado el número de neuronas para cada variable de acuerdo con el siguiente detalle:

- Caudal, 3 neuronas
- Nivel, 2 neuronas
- Potencia a producir, 2 neuronas
- Número de capas oculta de la RNA, 2

Para ejemplificar el armado de las ventanas (valores) que ingresaran a la RNA y de acuerdo con el análisis del PDS anteriormente descrito, se presenta el siguiente modelo en el cual se asignan valores a cada variable, un vector para cada una, como se visualiza en la figura 4-3.

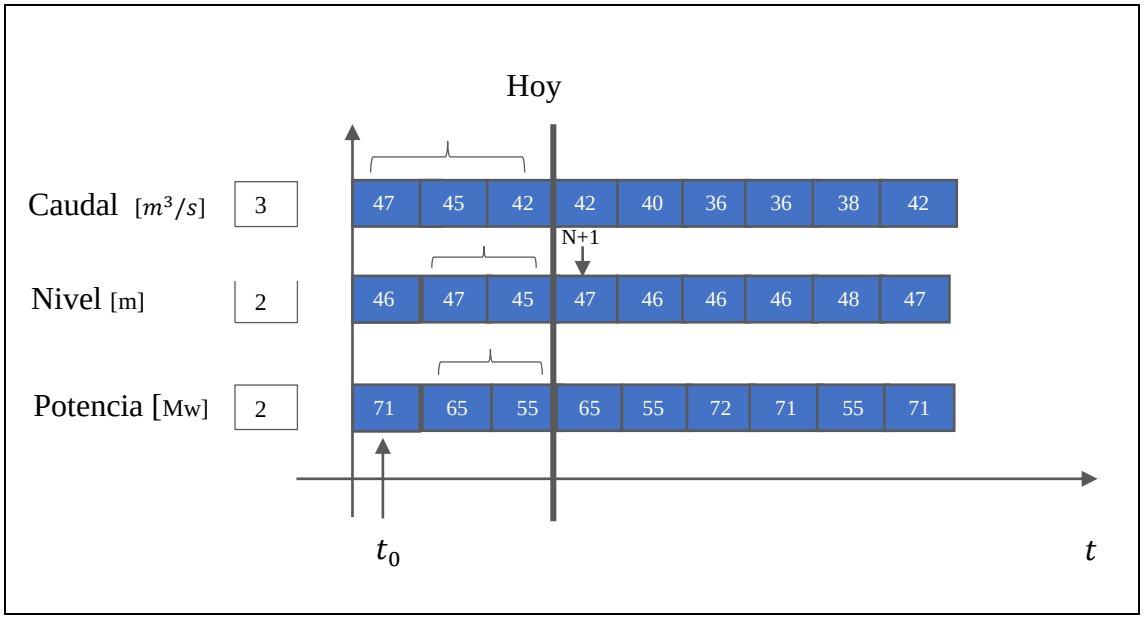


Figura 4-3: Valores por variable.
Realizado por: Edison Chafla, 2019.

De acuerdo con la figura 4-3, el número de datos para cada variable es 3, 2, 2 para el caudal, nivel y potencia respectivamente, la ventana resultante de acuerdo con estos parámetros iniciándolos en el t_0 (tiempo 0), es la siguiente:

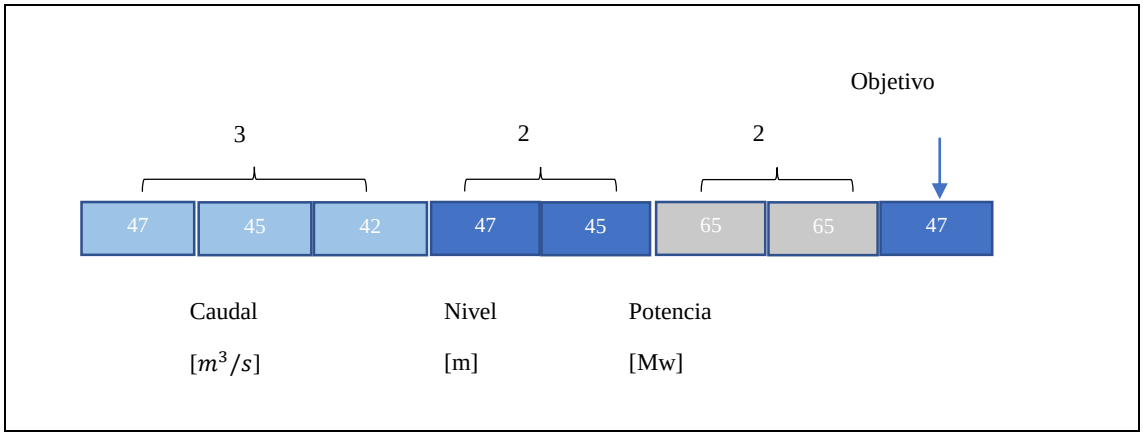


Figura 5-3: Armado de ventanas.
Realizado por: Edison Chafla, 2019.

3.1.7. Información para entrenamiento y validación

La información utilizada para el entrenamiento de las RNA, fueron escogidas de años al azar, revisando el historial de la base de datos, de acuerdo con el siguiente detalle:

Datos para entrenamiento: 01/01/2007 hasta el 31/12/2009.
Datos para la validación: 01/01/2010 hasta el 31/12/2012.

3.2. Determinación de los algoritmos

3.2.1. Parámetros de entrenamiento

El ingreso de datos al sistema se lo realizó por medio de la implementación de matrices, se creó la matriz de *entradas* y *objetivo*, de acuerdo con el ejemplo que se presenta en la figura 6-3, la matriz de entradas contiene los valores de las variables de nivel, caudal y potencia, en el instante t_0, por otro lado, la matriz objetivo contiene la información de la variable nivel *(N+1)* adelantada en una hora.

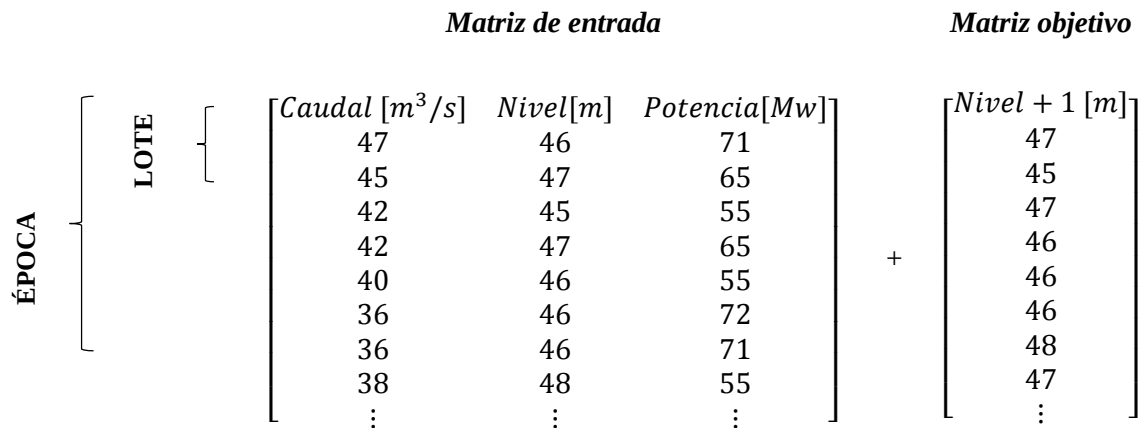


Figura 6-3: Armado de matrices.
Realizado por: Edison Chafla, 2019.

El aprendizaje se lo realiza de acuerdo con la presentación repetida de un conjunto de datos, cuando pasamos por la RNA todo el conjunto de datos lo denominamos época (epoch en inglés), el entrenamiento consiste en repetir época tras época hasta conseguir que los pesos sinápticos estabilicen los errores de entrenamiento y validación. Existen dos modos de actualización de pesos, el modo secuencial y el modo batch (lote en inglés), en el primero los pesos sinápticos se actualizan tras presentarse cada ejemplo de entrenamiento (batch), es decir se producen N

actualizaciones de pesos sinápticos durante cada época, en el segundo caso se produce una sola actualización durante una época (Bertona, 2005).

Las *iteraciones o número de entrenamientos* es la cantidad de veces que una época pasa por la red neuronal, el número de esta depende básicamente del error de entrenamiento y del error de validación, cuando estos valores son cercanos o se aproximan a 10^{-3}, y la varianza y desviación estándar de estos tienden a ser valores pequeños podemos concluir que el número de entrenamientos es aceptable.

Los valores de *época, lote e iteraciones* utilizados para el entrenamiento de las RNAs fueron los siguientes:

Tabla 3-3: Parámetros de entrenamiento.

Parámetros	**Valores**
Época (epoch)	15000
Lote (batch)	128
Iteraciones	5

Realizado por: Edison Chafla, 2019.

3.2.2. Modelo CLP

EL modelo CLP (por sus siglas en inglés, Close-Loop Prediction), o predicción en circuito cerrado, es un esquema en el cual la salida es retroalimentada a una de sus entradas, para nuestro caso, la salida que es realimentada es la variable de nivel *N+1(valor de predicción)*, en otras palabras, es el modelo MISO con realimentación a su entrada, como de detalla en la figura 7.3 (Asqui & Hernánez, 2017).

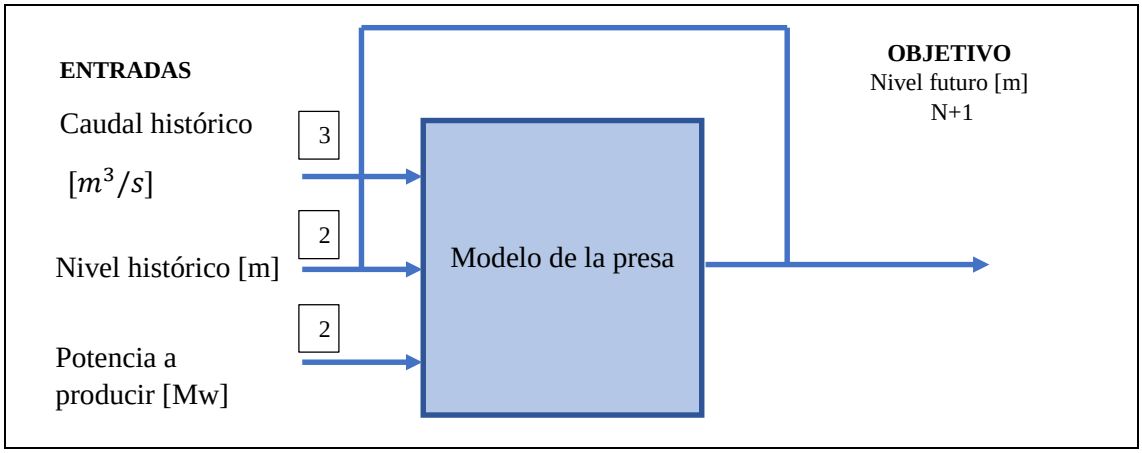


Figura 7-3: Modelo CLP.
Realizado por: Edison Chafla, 2019.

Una vez obtenida la RNA entrenada, para su validación se alimenta sus entradas con información de otro período o fecha, como respuesta entrega una predicción con un umbral de una hora de la variable de nivel (*N+1*), esta señal es realimentada al sistema. Este ciclo se repite de acuerdo con los umbrales de predicción que se desee analizar, los utilizados para esta investigación fueron de 4, 8, 24 y 48 horas de predicción.

3.2.3. Librería Keras

Keras es una librería de Python, en la cual se pueden realizar la creación de modelos de *Deep Learnig,* utlizando backends como TensorFlow, Theano o CNTK (Chollet, 2015b).

3.2.3.1. *Características básicas.*

Dentro de las principales tenemos:

- Es una librería escrita en Python.
- Diseñada para hacer experimentos con RNAs.
- Permite la creación de prototipos fácil y rápida (a través de la facilidad de uso, la modularidad y la extensibilidad).
- Admite redes convolucionales y redes recurrentes, así como combinaciones de las dos.
- Se ejecuta sin problemas en la CPU y la GPU.

3.2.3.2. *Definición del modelo*

La estructura de datos principal en Keras es un modelo(*model*), que permite su organización por medio de capas, ya que los modelos de Keras son una secuencia de capas. La clase keras.models.Sequential es una envoltura para el modelo de red neuronal (Torres, 2017).

```
#Codigo Creación Modelo
from keras.models import Sequential
model = Sequential()
```

Existen capas totalmente conectadas *(full connected)*, capas de agrupamiento *(max pool)*, capas de activación, etc. Para el entrenamiento de las RNAs utilizamos las primeras, para el ejemplo de entrenamiento descrito a continuación, una vez creado el modelo *(sequencial)*, definimos el número de capas y su forma de conexión, en este caso son totalmente conectadas por medio del método *Dense*, para añadir más capas a la RNA utilizamos el método *model.add.* el número de

neuronas de las capas ocultas corresponden a la temporada climática *invernal fuerte*. Adicional se utiliza la función de activación sigmoidal para cada neurona (Torres, 2017).

```
#Codigo Creacion Modelo
model = Sequential()
neuronasEntradas = 7
#Dense para crear capas full conectadas
neuronas=50
neuronas1 =50
model.add(Dense(neuronas, input_dim=neuronasEntradas, activation='sigmoid'))
model.add(Dense(neuronas1, activation='sigmoid'))
model.add(Dense(1, activation='sigmoid'))
```

Keras deducirá de manera automática la forma de todas las capas después de la primera capa. La primera capa model.add(Dense(neuronas,input_dim=neuronasEntradas, activation='sigmoid'))), establece la dimensión de entrada en 7 (lo que significa que los datos de entrada tiene 7 dimensiones). La segunda capa toma la salida de la primera y la enlaza a la última capa, para nuestro ejemplo la salida tiene dimensión 1, por lo tanto, cumple con la topología del modelo MISO(Torres, 2017).

3.2.3.3. *Aprendizaje*

Al tener el modelo de la RNA construido, procedemos a configurar el proceso de aprendizaje con el método .compile(). La combinación del modelo utiliza las librerías numéricas de TensorFlow, este backend selecciona de forma automática la mejor forma para representar la red para el entrenamiento.

Para compilar se debe especificar algunas propiedades adicionales requeridas para el entrenamiento de la RNA, como es la función loss (función de pérdida), el optimizador utilizado para buscar los diferentes pesos en la RNA y cualquier medida opcional que se desee registrar durante el entrenamiento. En este caso se utiliza el algoritmo Nadam y la función de pérdida es el erro medio cuadrático (mean squared error) (Torres, 2017).

```
optimizador='Nadam'
#si se desea graficar la accuracy
model.compile(optimizer=optimizador,loss='mean_squared_error',metrics=['accuracy'])
```

```
#si no se desea dibujar la accuracy
model.compile(optimizer=optimizador,loss='mean_squared_error')
```

3.2.3.4. *Entrenamiento*

Una vez definido y compilado nuestro modelo está listo para ser procesado, por lo tanto, procedemos a entrenarlo con un conjunto de datos. Además podemos entrenar o ajustar el modelo con los datos cargados llamando a la función .fit() del modelo (Torres, 2017).

El proceso de entrenamiento se ejecutará para un número fijo de iteraciones a través del conjunto de datos denominado *epochs* (ver Tabla 3-3, Parámetros de entrenamiento). También definimos el tamaño del lote del conjunto de datos, antes que se realice una actualización de pesos en la RNA mediante el argumento *batch_size* (Torres, 2017).

Un *Callbacks*, es un conjunto de métodos que se aplican en diferentes etapas del procedimiento de entrenamiento, para este entrenamiento se utilizó el callback *earlystop,* el cual permite parar el entrenamiento cuando el error tiende a estabilizarse, esto se define por medio del parámetro *patience,* otro callback utilizado fue el *checkpoint*, el cual guarda la mejor red en cada período. La red de almacena localmente en un archivo *.hdf5. Finalmente utilizamos el callbakc *history,* este se aplica a todos los algoritmos de entrenamiento de manera automática, este objeto devuelve el método *fit ()* de los modelos (Chollet, 2015a).

```
#Codigo Entrenar Modelo
earlystop = EarlyStopping(monitor='val_loss', patience=9000, verbose=0, mode='min')
#grabar el mejor modelo de este entrenamiento
filepath = "C:/PPCHEC2/Entrenadores/RedModelo/weights-{epoch:02d}-{val_loss:.5f}.hdf5"
#guarda un checkpoint de la mejor red que había en cada período}
checkpoint = ModelCheckpoint(filepath, monitor='val_loss', verbose=0, save_best_only=True,
save_weights_only=False, mode='min', period=5000)
callbacks_list = [earlystop,checkpoint]
#selección automática de conjunto de validación
model.fit(enEntrenamiento, objEntrenamiento,validation_split=0.33, epochs=epocas,
batch_size=batch, verbose=2)
#selección manual de conjunto de validación
history = model.fit(enEntrenamiento, objEntrenamiento,
validation_data=(enValidacion,objValidacion), epochs=epocas,
batch_size=batch, verbose=2, callbacks=callbacks_list)
```

3.2.3.5. *Evaluación del modelo*

Una vez entrenada nuestra red neuronal procedemos a evaluarla su eficiencia por medio de la función *.evaluated (),* utilizando un conjunto de datos apropiado, como se detalla en el siguiente código.

```
loss_and_metrics = model.evaluate(x_test, y_test)
```

3.2.3.6. *Generación de predicciones*

Con el objeto de generar predicciones sobre nuevos datos, se puede utilizar la función *.predict(),*como se detalla en el siguiente código

```
classes = model.predict(x_test, batch_size=128)
```

3.2.4. Generación de matrices para la predicción

Para la etapa de la validación del error de predicción de nivel de agua de embalse de una RNA entrenada (archivos *.hdf5), es necesario generar una matriz que contenga los datos de las variables de nivel, caudal y potencia, como se observa en la figura 7-3 Modelo CLP.

Como primer paso escogemos una fecha a partir de la cual iniciaremos la prueba, para el ejemplo hemos seleccionado el 2007/01/01, como se observa en la figura 8.3, antes de esta fecha se encuentran los datos del *ayer,* y después de estos están los datos *futuros o umbral,* la cantidad de datos del umbral dependerá de las horas para las cuales se requiere realizar el análisis de la predicción, para el ejemplo se ha seleccionado un umbral de 4 horas.

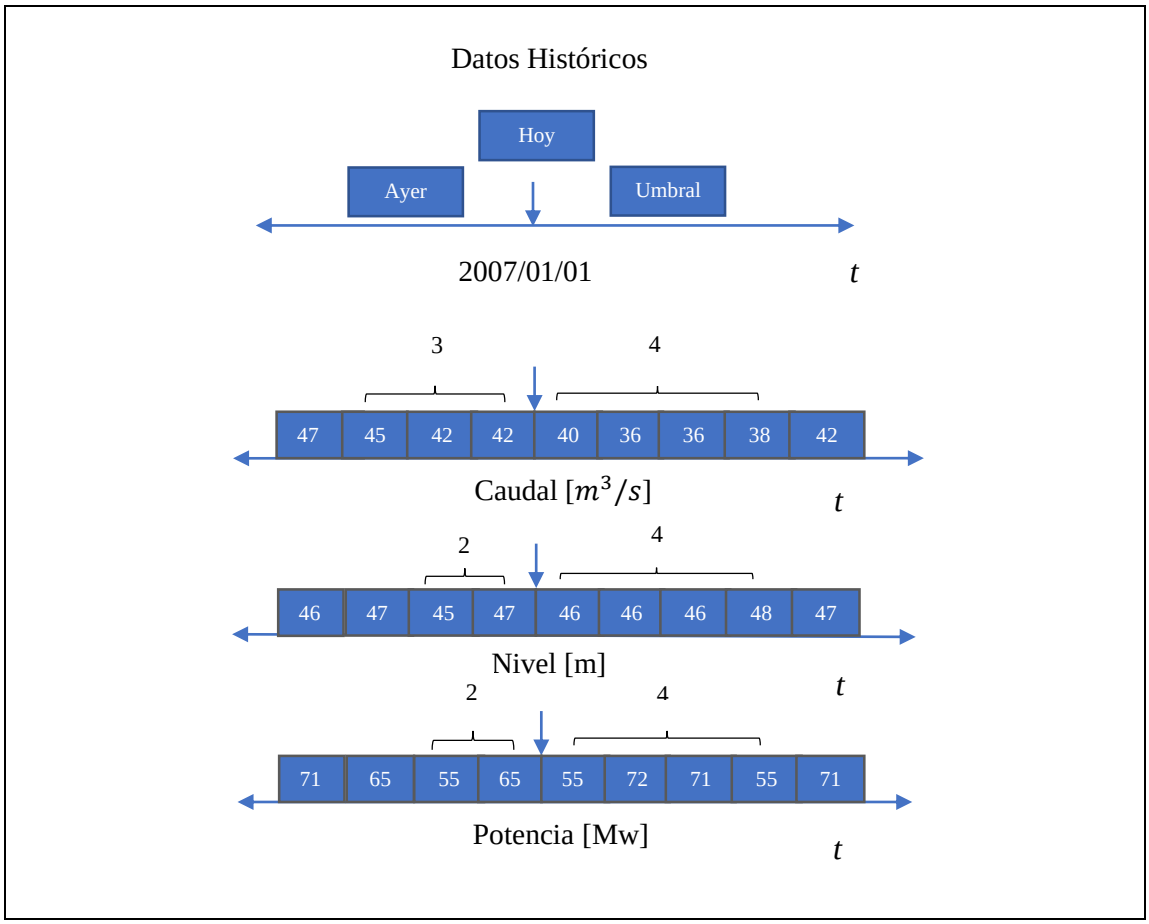


Figura 8-3: Validación de una RNA entrenada para un umbral de 4 horas.
Realizado por: Edison Chafla, 2019.

El número de neuronas de entrada es 7 (ver figura 4-3. Valores por variable), para llenar la matriz se selecciona el valor de la ventana más grande de las variables de entrada (caudal: 3, nivel:2, potencia:2), es decir 3 valores para cada variable. Tanto los valores de los datos del *ayer y umbral* son extraídos de la base de datos. La ejemplificación de lo detallando anteriormente se puede observar en la figura 9-3. Validación de una RNA entrenada.

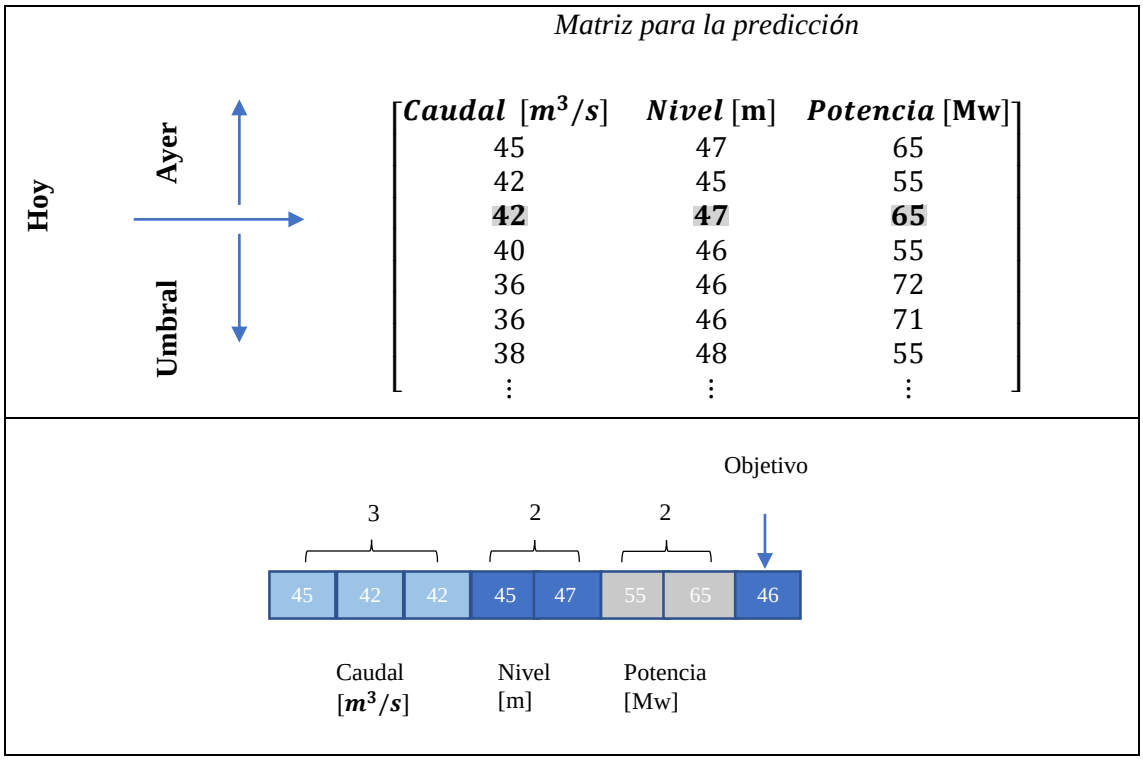


Figura 9-3: Matriz de validación de una RNA Entrenada.
Realizado por: Edison Chafla, 2019.

3.2.5. Programación con hilos

Threading es la técnica que en programación permite que una aplicación ejecute de manera simultánea varias operaciones al mismo tiempo, el hilo o subproceso es cada flujo de ejecución que genera durante el procesamiento, pudiendo realizar o no una misma tarea, es similar a ejecutar varios programas diferentes al mismo tiempo. En Python, el módulo *threading* hace posible la programación en hilos. Dentro de las principales ventajas del uso de esta técnica tenemos (Ceballos, 2017).

- Los hilos en ejecución de un proceso comparten el mismo espacio de datos que el hilo principal y pueden, por tanto, tener acceso a la misma información o comunicarse entre sí más fácilmente que si estuvieran en procesos separados.
- Ejecutar un proceso de varios hilos suele requerir menos recursos de memoria que ejecutar lo equivalente en procesos separados.
- Permite simplificar el diseño de las aplicaciones que necesitan ejecutar varias operaciones concurrentemente.

Para cada hilo de un proceso existe un puntero que realiza el seguimiento de las instrucciones que se ejecutan en cada momento. Además, la ejecución de un hilo se puede detener temporalmente

o de manera indefinida. En general, un proceso sigue en ejecución cuando al menos uno de sus hilos permanece activo, es decir, cuando el último hilo concluye su cometido, termina el proceso, liberándose en ese momento todos los recursos utilizados (Ceballos, 2017).

En Python un objeto Thread, representa una determinada operación que se ejecuta como un subproceso independiente, es decir, representa a un hilo. Hay dos formas de definir un hilo: la primera, consiste en pasar al método constructor un objeto invocable, como una función, que es llamada cuando se inicia la ejecución del hilo, la segunda, radica en crear una subclase de Thread en la que se reescribe el método run() y/o el constructor __init__() (Ceballos, 2017).

Para el siguiente ejemplo se crean tres hilos (*PruebaMISO*, *PruebaSISO*, *PruebaMISOSISO*), heredan de la clase abstracta *Prueba*, los métodos están implementados en las clases *PruebaMISO*, *PruebaSISO*, *PruebaMISOSISO.*

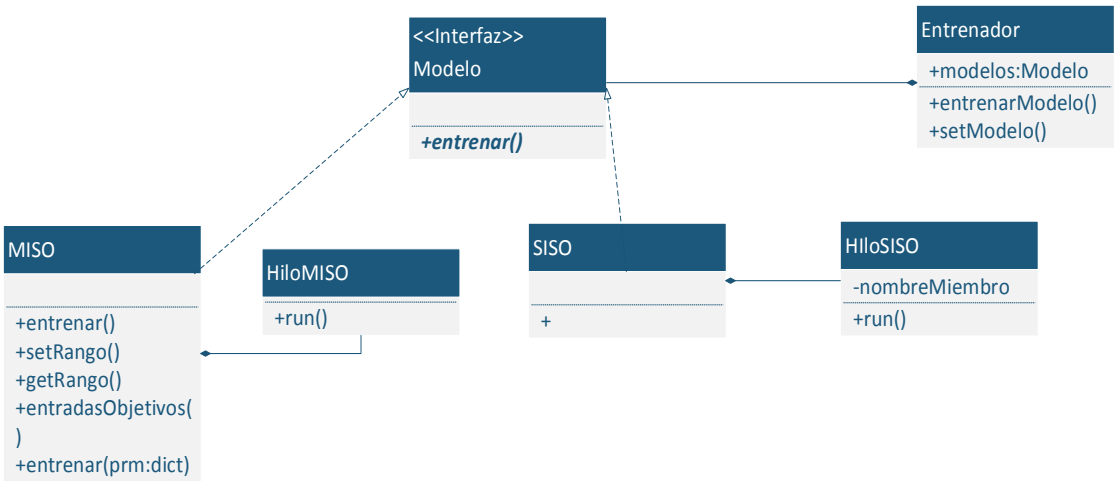


Figura 10-3: Implementación de multithreading.
Realizado por: Edison Chafla, 2019.

De acuerdo con el esquema presentado en la figura 10-3, para que entre en ejecución el hilo que contiene a la clase *PruebaMISO(),* es necesario que dentro de su código se incluya el método *.start(),* de esta manera mientras se ejecuta el código del hilo *PruebaMISO*, se continuarán ejecutando la aplicación sin tener el problema de que el sistema operativo termine el programa por pensar que este último no responde.

```
#CódigoMultithreading
import threading

class Prueba(AccionesComunes):
```

```
    """clase abstracta la prueba particular implementa el método probar.
    Operaciones comunes para las pruebas de MISO, SISO, SIMI"""
    def __init__(self):
        super().__init__()
    def probar(self, prmModelo={}, prmIHM={}, IHM=QWidget, filaRMSE=[0]):
        pass
class PruebaMISO(Prueba):
    """Realiza pruebas al MISO para el umbral dado dentro de prmIHM."""
    def __init__(self):
        super().__init__()
        self.pathModelo = ''
    def probar(self, prmModelo={}, prmIHM={}, IHM=QWidget, filaRMSE=[0]):
        hilo = HiloPruebaMISO(prmModelo, prmIHM, IHM, filaRMSE)
        hilo.start()
class HiloPruebaMISO(threading.Thread, AccionesComunes):
    def __init__(self, prmModelo={}, prmIHM={}, IHM=QWidget, filaRMSE=[0]):
        super(HiloPruebaMISO, self).__init__()
        self.prmModelo = prmModelo
        self.prmIHM = prmIHM
        self.IHM = IHM
        self.pathModelo = ''
        self.filaRMSE = filaRMSE
```

3.3. Implementación de la plataforma.

La plataforma de software fue desarrollada en el lenguaje de programación Python en su versión 3.6, para las interfases de usuario se utilizó PyQt en su versión 5, PyQt5 es un método de la biblioteca gráfica Qt para el lenguaje de programación Python. Los datos fueron almacenados en el gestor de base de datos PostgreSQL.

Para el entrenamiento de las RNAs utilizamos los algoritmos de entrenamiento de la librería Keras de Python, esta es una biblioteca a nivel de modelo, que proporciona bloques de construcción de alto nivel para desarrollar modelos de aprendizaje profundo. No maneja operaciones de bajo nivel, como productos tensores, convoluciones, etc. En cambio, se basa en una biblioteca de manipulación de tensores especializada y bien optimizada para hacerlo, que actúa como el "motor de back-end" de Keras. En lugar de elegir una sola biblioteca de tensores y hacer que la implementación de Keras esté ligada a esa biblioteca, Keras maneja el problema de una manera

modular, y varios motores back-end diferentes pueden conectarse sin problemas a Keras (Chollet, 2007).

En este momento, Keras tiene tres implementaciones de backend disponibles: el backend TensorFlow , el backend Theano y el backend CNTK .

Keras puede funcionar sin problemas en tanto CPU o GPU. Cuando se ejecuta en la CPU, TensorFlow utiliza una biblioteca de bajo nivel para las operaciones de tensor llamada Eigen. En GPU, TensorFlow utiliza la biblioteca de operaciones de aprendizaje profundo bien optimizadas llamada biblioteca NVIDIA CUDA Deep Neural Network (cuDNN)(Chollet, 2007).

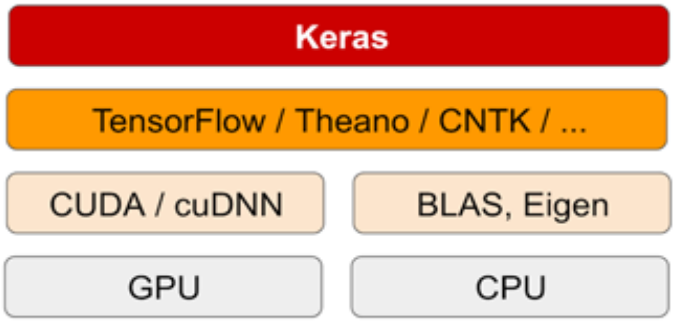


Figura 11-3: Aprendizaje profundo estructura hardware-software.
Fuente: ("Deep Learning with Python", 2018)

3.3.1. Configuración de la GPU, Tensorflow & Keras

A continuación, se detallará los pasos para la configuración de la estructura hardware y software utilizada.

3.3.1.1. Ambiente

El computador utilizado tiene las siguientes características:

- Windows 10 Home
- NVIDIA GeForce GTX 1050Ti 4 GB GDDR5
- Core i7-7700 HQ
- 12GB DDR4 RAM

3.3.1.2. Configuración del back-end para TensorFlow

Es necesaria la descarga de la última versión de Anaconda, esta es una plataforma de software moderno y dinámica que permite el desarrollo y automatización del aprendizaje profundo. Es la forma fácil y rápida de trabajar con Python y sus librerías para el aprendizaje automático en Linux, Windows y Mac, en una sola máquina.

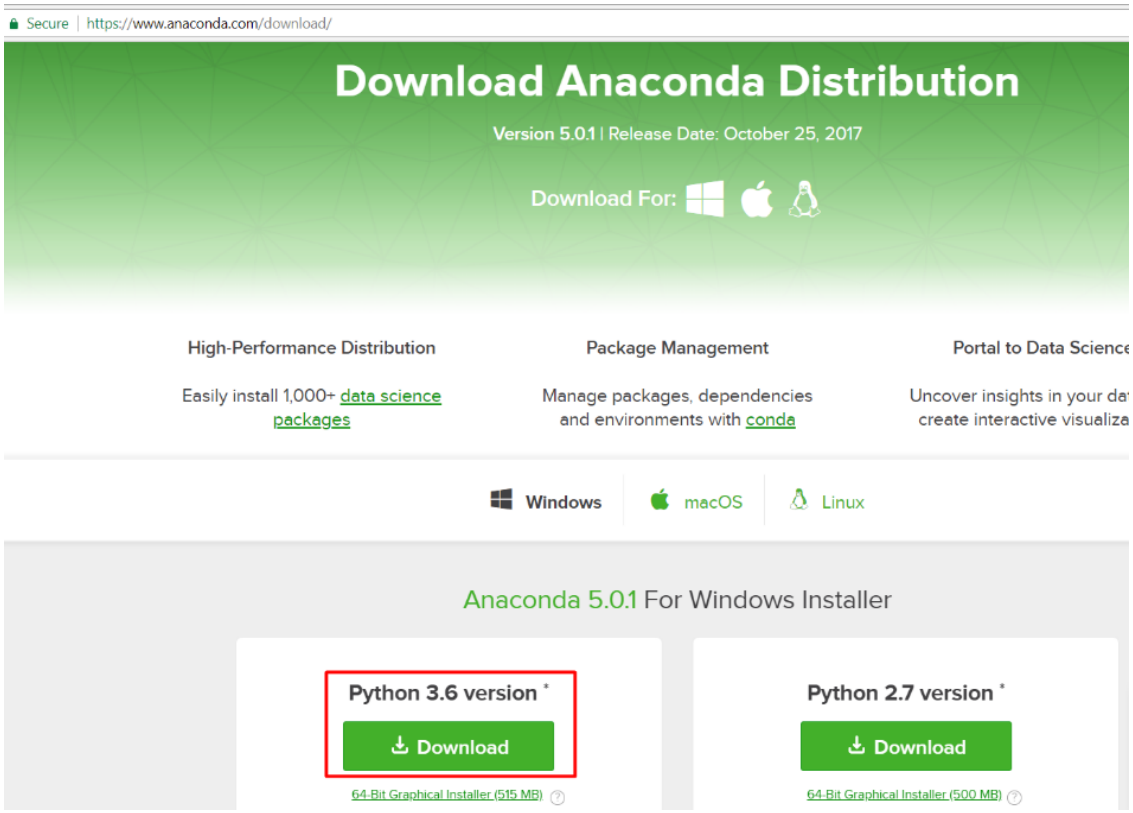


Figura 12-3: Instalación de Anaconda
Fuente: (“ www.anaconda.com”, 2018)

3.3.1.3. Instalación y descarga de Visual Studio 2017

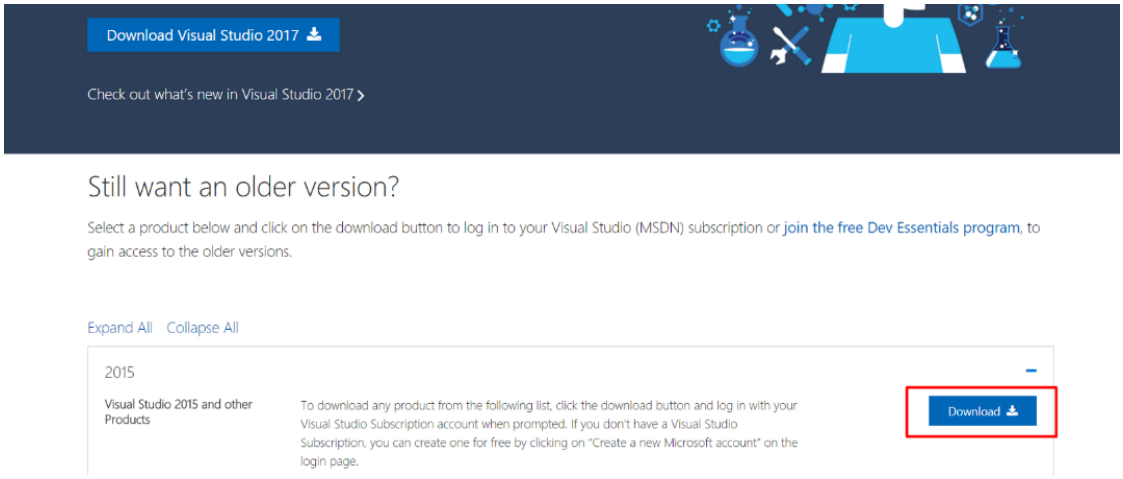


Figura 13-3: Visual Studio
Fuente: (“ www.visualstudio.com”, 2018)

3.3.1.4. Cuda Toolkit 8.0

NVIDIA® CUDA® toolkit proporciona un entorno de desarrollo para crear aplicaciones por medio de la GPU.

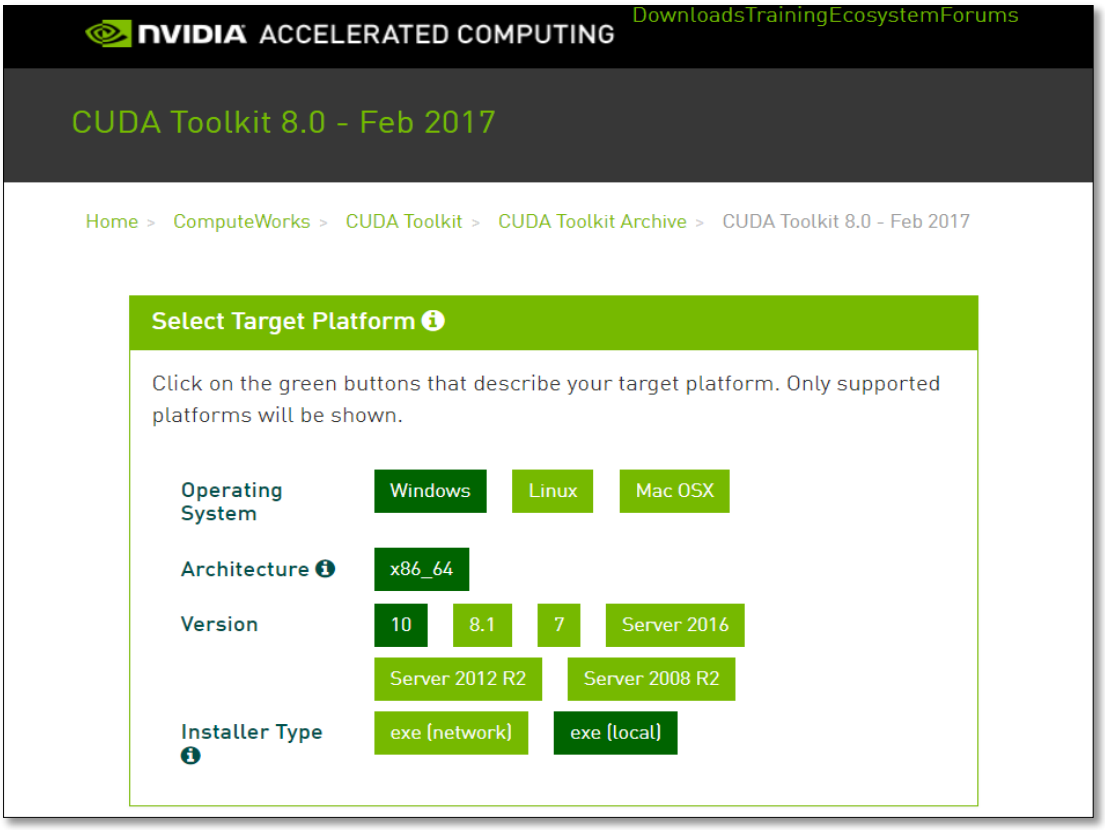


Figura 14-3: Cuda Toolkit.
Fuente: (“www.nvidia.com”, 2018)

3.3.1.5. *cuDNN v 6.0 para CUDA 8.0.*

La biblioteca NVIDIA CUDA® Deep Neural Network (cuDNN) es una biblioteca acelerada por GPU para redes neuronales profundas .cuDNN proporciona implementaciones altamente sintonizadas para rutinas estándar tales como convolución hacia adelante y hacia atrás, puesta en común, normalización y capas de activación.

3.3.1.6. *Instalación de TensorFlow.*

En Anaconda por medio de un Promt, ingresamos el siguiente comando:

Conda create –name tensorflow –clone root

El comando *-clone root*, heredará las bibliotecas del entorno de Python a TensorFlow.

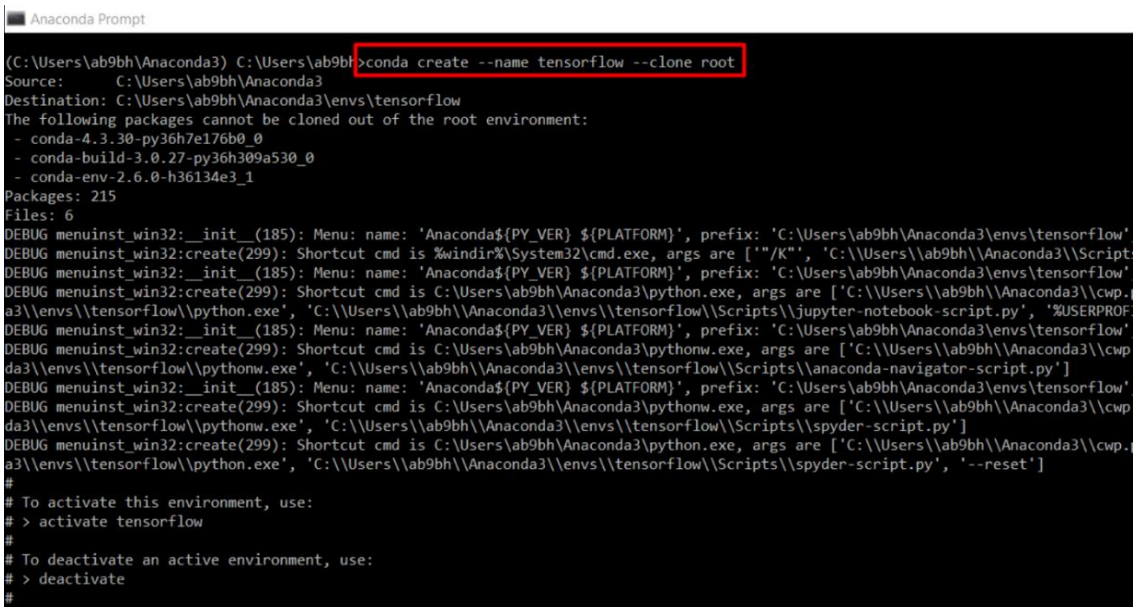


Figura 15-3: Instalación de TensorFlow.
Realizado por: Edison Chafla, 2019.

3.3.1.7. *Instalación de TensorFlow para GPU.*

Ingresamos los comandos:

activate tensorflow

pip –ignore-installed –upgrade tensorflow-gpu

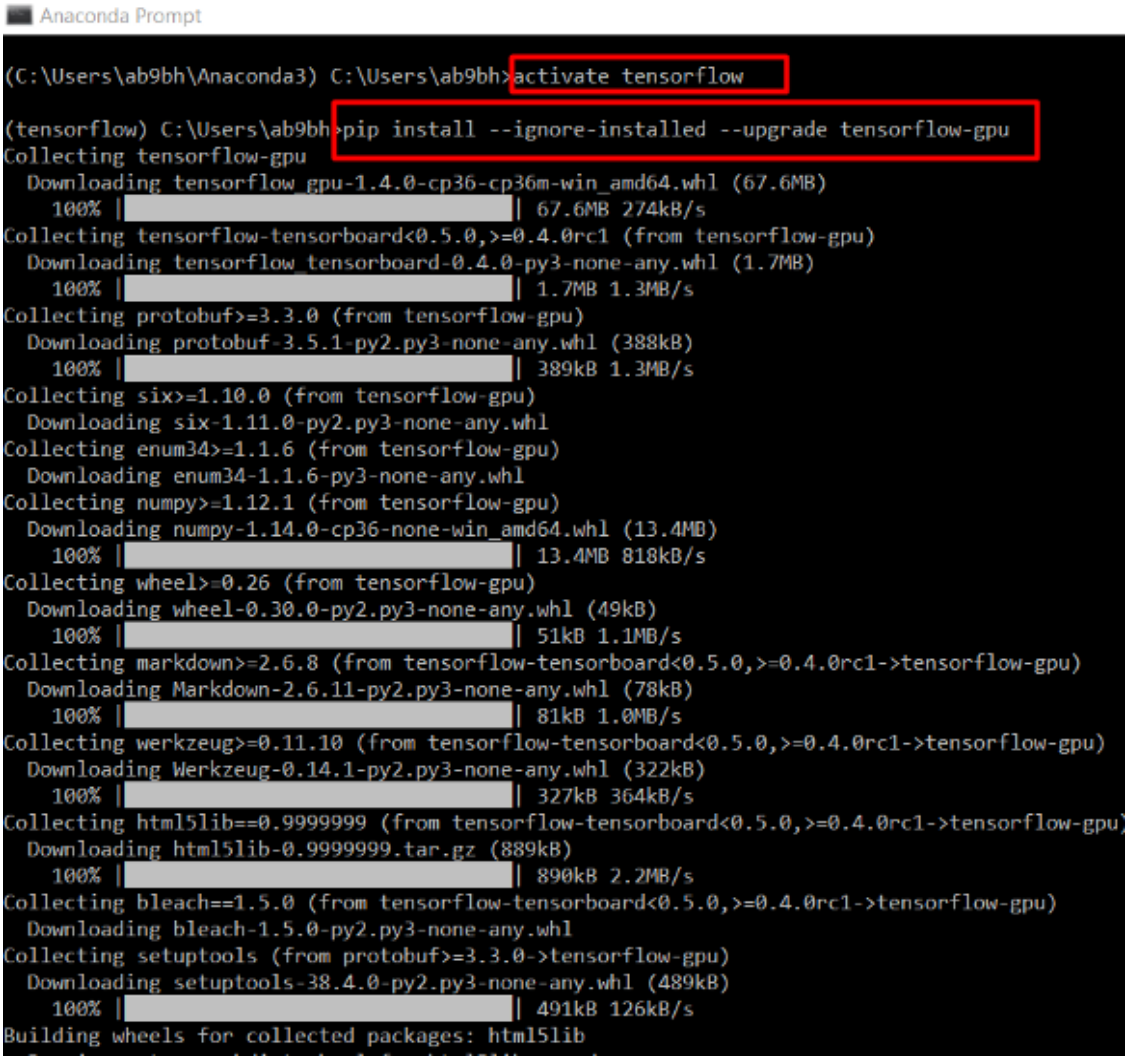


Figura 16-3: Instalación de TensorFlow para GPU.
Realizado por: Edison Chafla, 2019.

3.3.1.8. Instalación de Keras.

En Anaconda por medio de un promt, abrimos el entorno de Tensorflow usando *´activate tensorflow´*, e ingresamos el siguiente comando:

conda install keras

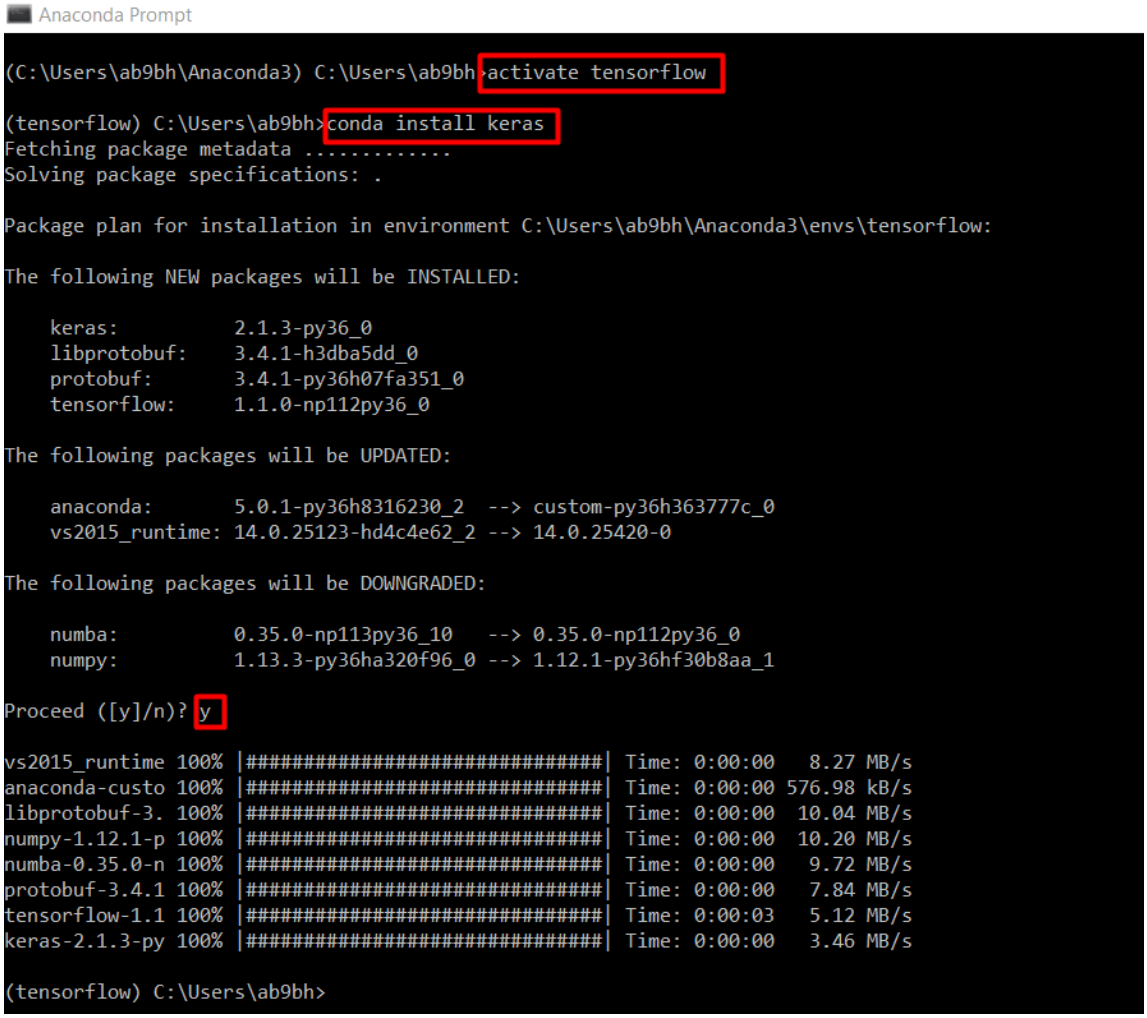


Figura 17-3: Instalación de Keras.
Realizado por: Edison Chafla, 2019.

3.3.2. Plataforma

La plataforma de software consta de 5 módulos, que se detallan a continuación:

- Usuario, contiene los métodos para el control y autenticación de usuarios que ingresan al sistema.
- BDD, contiene los métodos que me permiten acceder a datos en la BDD, para grabarlos o modificarlos.
- Datos, contiene los métodos que analizan el PDS de las variables que ingresan a la RNA.
- Entrenadores, en este módulo se encuentran todos los métodos que me permiten entrenar las RNAs.
- Pruebas, contiene los métodos para probar las RNAs entrenadas y prueba los modelos para la presa.
- LanzIHMRNA2, permite establecer el método de comunicación entre los distintos módulos.

3.4. Entrenamiento de los modelos

Una vez entrenado el modelo o RNA, con los distintos algoritmos u optimizadores de la librería Keras de Python, es necesario realizar un análisis y evaluación de estos, para esto estudiamos comportamiento del error de entrenamiento, error de validación y error de predicción de nivel de agua de embalse con cada uno de los modelos y de esta manera poder reconocer cual algoritmo presenta un óptimo rendimiento. Las variables utilizadas para este análisis son las siguientes:

Tabla 4-3: Variables independientes y dependientes.

Variable Independiente	**Concepto**
Número de algoritmos de entrenamiento	Mide el número de algoritmos de entrenamiento que sirven para realizar el modelamiento de una serie temporal proporcionados por la herramienta Open Source Keras.
Tiempo de aprendizaje de los algoritmos de entrenamiento	Es el tiempo que toma cada algoritmo de entrenamiento de una RNA, para el modelado de represas mediante la utilización de un computador y una GPU.
Variable Dependiente	**Concepto**
Error de Predicción de Nivel	Diferencia entre el valor real de nivel futuro y el valor futuro predicho por la red neuronal artificial.
Error de entrenamiento	Es la diferencia entre el valor entregado por la RNA y los valores reales escogidos para el entrenamiento de esta.
Error de Validación	Es la diferencia entre el valor entregado por la RNA y los valores reales que no están dentro del conjunto de valores de entrenamiento.

Realizado por: Edison Chafla, 2019.

El proceso de validación realizado consta de las siguientes etapas:

- Determinación del número de épocas e iteraciones.
- Estabilización del error de entrenamiento y validación.
- Entrenamiento de los algoritmos.
- Comparación de los algoritmos.
- Validación del modelo.

3.4.1. Determinación del número de épocas e iteraciones.

Como primera parte de esta validación fue necesario analizar el comportamiento del error de entrenamiento y validación con cada uno de los algoritmos de entrenamiento, para esto establecimos igual número de parámetros y valores para todos. En la tabla 5-3, se presentan estos valores.

Tabla 5-3: Parámetros de entrenamiento.

Parámetros	**Valores**
Epoch	15000
Batch	128
Número de capas ocultas	2
Patience	5000
Period	5000

Realizado por: Edison Chafla, 2019.

Estas pruebas fueron realizadas para las tres temporadas climáticas (ver Tabla 2-3), que se describen en el apartado 3.1.3 de este documento, los parámetros analizados fueron el error de entrenamiento, error de validación y tiempo de entrenamiento. Cabe indicar que se registra únicamente el valor mínimo de cada error producido durante las 15000 épocas, para cada uno de los algoritmos de entrenamiento, esto con el afán de determinar el número de épocas adecuadas para el entrenamiento. Los resultados obtenidos se presentan en la tabla 6-3, tabla 7-3 y tabla 8-3

Para la representación de estos resultados se encuentran en los gráficos 1-3, 2-3 y 3-3, se realizó un escalado de los mismo, esto con el objeto de poder representar en un solo gráfico los tres valores analizados, ya que los valores de la variable tiempo de entrenamiento son muy grandes respecto al valor de los errores.

Tabla 6-3: Prueba de entrenamiento, temporada invernal fuerte.

Item	Algoritmo	Error de entrenamiento mínimo	Error de validación mínimo	Tiempo de entrenamiento Minutos [min]
1	SGD	1,18E-03	1,09E-03	88,48
2	RMSprop	2,75E-04	1,88E-04	64,04
3	Adagrad	3,71E-04	2,42E-04	105,40
4	Adadelta	3,45E-04	2,26E-04	78,87
5	Adam	2,65E-04	1,94E-04	87,50
6	Adamax	2,74E-04	1,97E-04	70,75
7	Nadam	2,26E-04	1,90E-04	67,33

Realizado por: Edison Chafla, 2019.

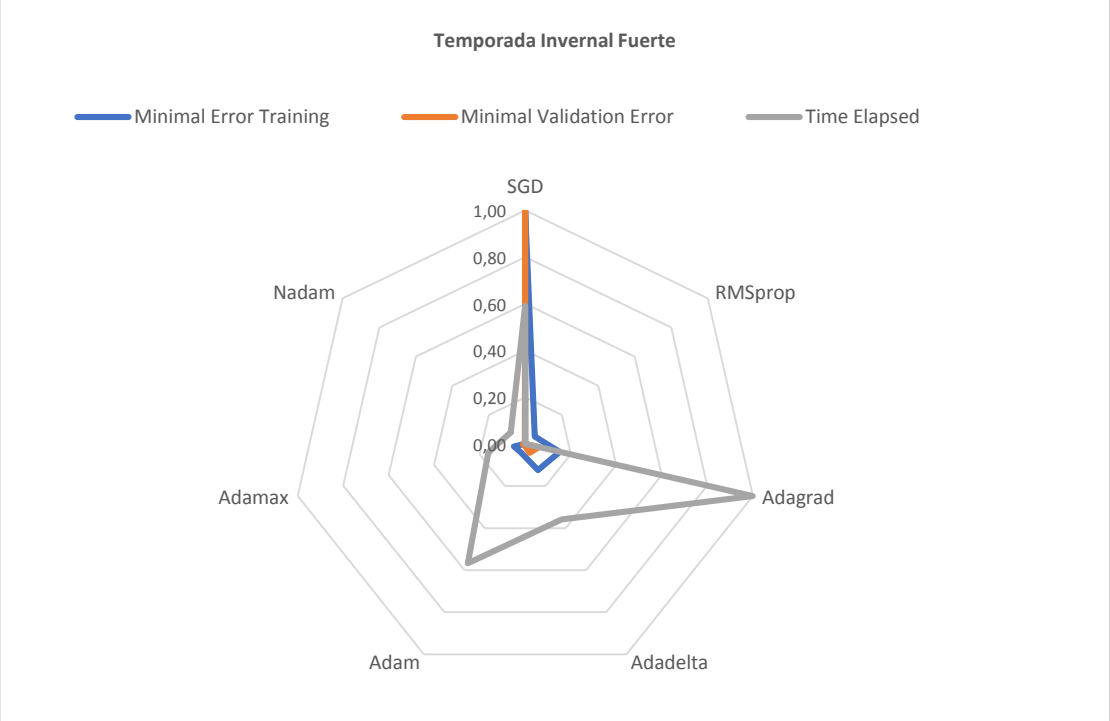


Gráfico 1-3: Temporada invernal fuerte, 15000 épocas.
Realizado por: Edison Chafla, 2019.

Tabla 7-3: Prueba de entrenamiento, temporada invernal ligera.

Item	Algoritmo	Error de entrenamiento mínimo	Error de validación mínimo	|Tiempo de entrenamiento Minutos [min]
1	SGD	7,69E-04	6,96E-04	55,68
2	RMSprop	2,13E-04	1,89E-04	53,81
3	Adagrad	2,64E-04	2,25E-04	53,53
4	Adadelta	2,62E-04	2,13E-04	74,02
5	Adam	2,13E-04	1,89E-04	78,52
6	Adamax	2,17E-04	1,85E-04	74,04
7	Nadam	2,03E-04	1,87E-04	55,19

Realizado por: Edison Chafla, 2019.

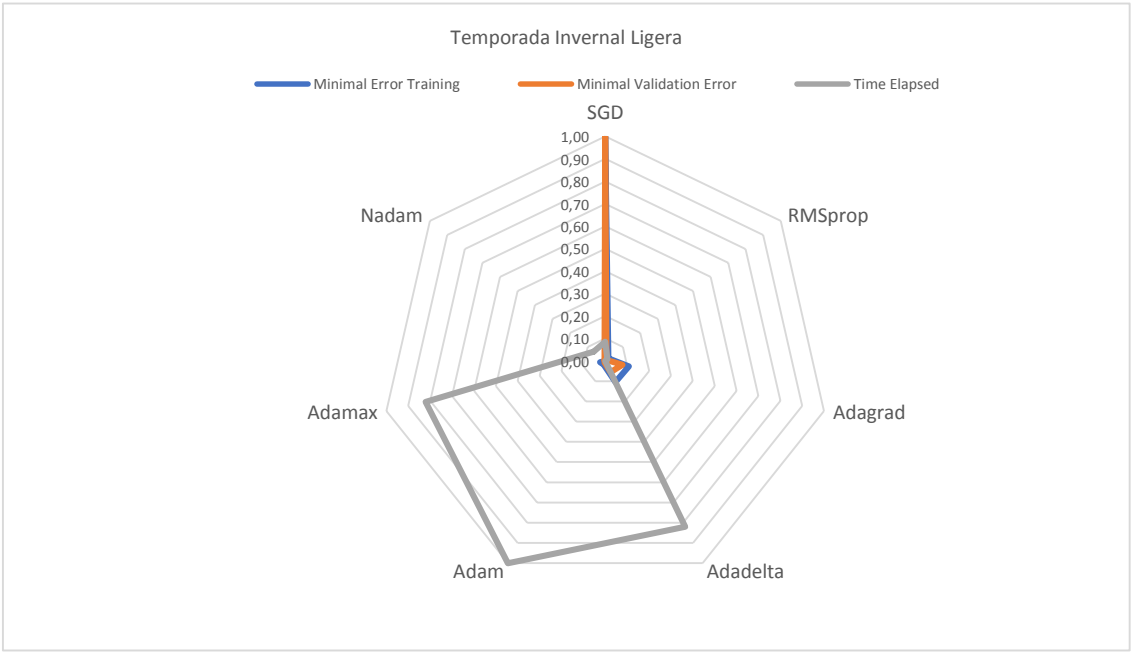


Gráfico 2-3: Temporada invernal ligera, 15000 épocas.
Realizado por: Edison Chafla, 2019.

Tabla 8-3: Prueba de entrenamiento, temporada de verano.

Item	Algoritmo	Error de entrenamiento mínimo	Error de validación mínimo	Tiempo de entrenamiento Minutos [min]
1	SGD	8,25E-04	7,59E-04	62,68
2	RMSprop	3,09E-04	2,69E-04	73,26
3	Adagrad	3,79E-04	3,14E-04	82,78
4	Adadelta	3,58E-04	2,88E-04	74,36
5	Adam	3,01E-04	2,75E-04	71,04
6	Adamax	3,10E-04	2,80E-04	58,71
7	Nadam	2,56E-04	2,74E-04	50,88

Realizado por: Edison Chafla, 2019.

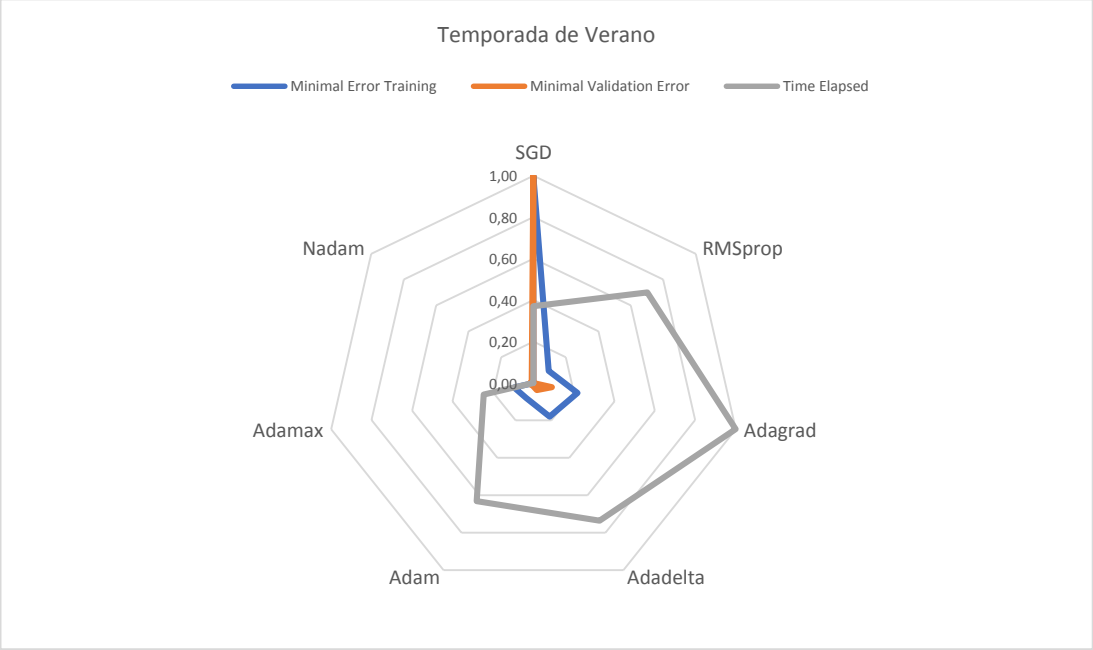


Gráfico 3-3: Temporada de verano, 15000 épocas.
Realizado por: Edison Chafla, 2019.

El tiempo total de entrenamiento de estos modelos se detallan en la tabla 9-3.

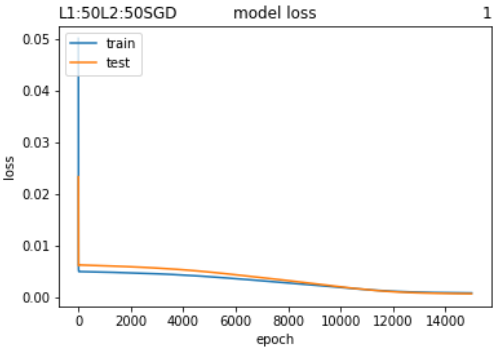
Tabla 9-3: Total de tiempo en las pruebas.

Item	**Algoritmo**	**Temporada invernal fuerte Minutos [min]**	**Temporada invernal ligera Minutos [min]**	**Temporada de verano Minutos [min]**	**Total Tiempo Minutos [min]**
1	SGD	88,48	55,68	62,68	206,84
2	RMSprop	64,04	53,81	73,26	191,11
3	Adagrad	105,40	53,53	82,78	241,71
4	Adadelta	78,87	74,02	74,36	227,25
5	Adam	87,50	78,52	71,04	237,06
6	Adamax	70,75	74,04	58,71	203,5
7	Nadam	67,33	55,19	50,88	173,4
				Total(minutos)	1480,87
				Total (horas)	24,7

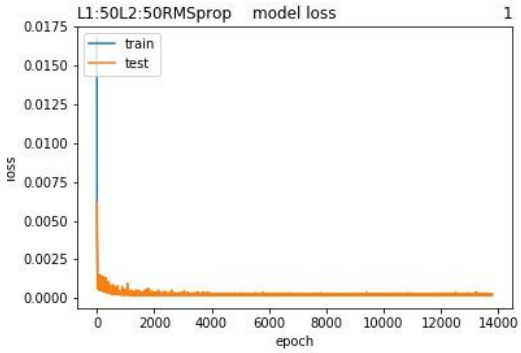
Realizado por: Edison Chafla, 2019.

Se realizó una inspección gráfica de las variables, error de entrenamiento y validación para determinar de manera aproximada la época a la cual, estas dos variables tienen a estabilizarse, su desempeño se muestra en el gráfico 4-3.

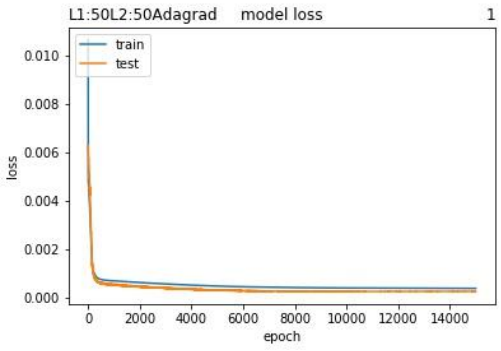
Ejemplo de un algoritmo de entrenamiento donde se determinó el número de épocas para el aprendizaje.



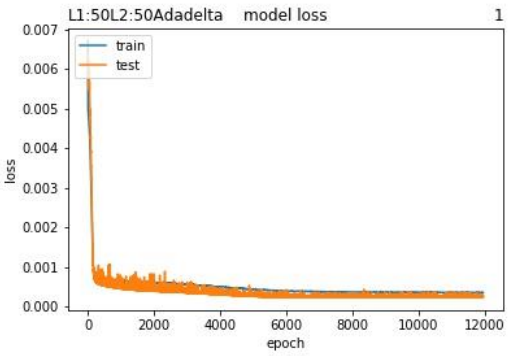
(a) SGD



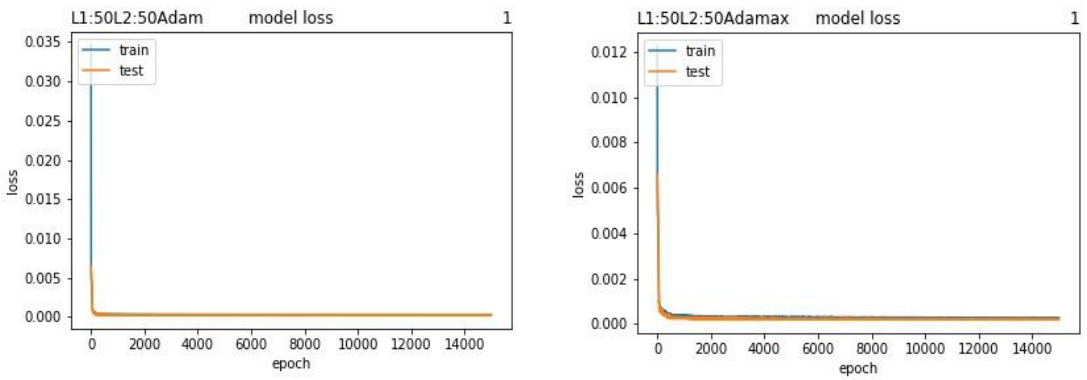
(b) RMSprop



(c) Adagrad

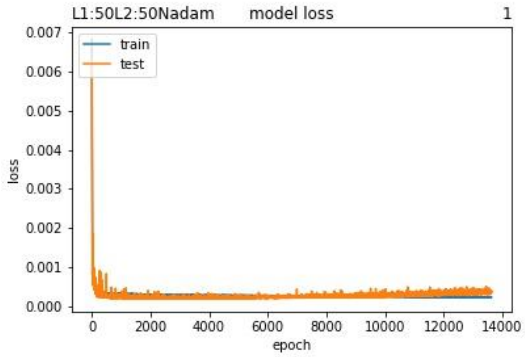


(d) Adadelta



(e) Adam

(f) Adadelta



(g) Nadam

Gráfico 4-3: Desempeño del error de entrenamiento y validación.
Realizado por: Edison Chafla, 2019.

De acuerdo con los resultados obtenidos sobre el número de épocas, tiempos de entrenamiento y la varianza, podemos determinar un valor estimado del número de épocas e iteraciones para el entrenamiento de los modelos, con el objetivo de reducir tiempos de entrenamiento y ahorrar los recursos computacionales disponibles, ya que como se puede observar estos dos parámetros tienden a estabilizarse en una época determinada, es decir estos valores se mantienen conforme el número de épocas aumenta, los valores obtenidos se detallan en la tabla 10-3.

Tabla 10-3: Parámetros para el entrenamiento del modelo.

Item	**Algoritmo**	**Épocas**	**Batch**	**Número de entrenamientos**
1	SGD	15000	128	5
2	RMSprop	2000	128	5
3	Adagrad	750	128	5
4	Adadelta	1000	128	5
5	Adam	750	128	5
6	Adamax	1000	128	5
7	Nadam	1000	128	5

Realizado por: Edison Chafla, 2019.

Es importante recalcar que, el realizar demasiadas veces un entrenamiento puede provocar que el modelo sufra un sobreajuste *(overfitting),* como se puede observar en el literal (g) del Gráfico 4-3. El valor del error de validación después de la época 10200 (aproximadamente), tienen a subir, entonces aprender casos particulares y es incapaz de reconocer nuevos datos. Por otro lado, en los entrenamientos también se puede presentar un subajuste (*underfitting),* este se produce cuando se carece de suficientes muestras de entrenamiento, entonces indica la imposibilidad de identificar u obtener resultados correctos (Villa, Velásquez, & Sánchez, 2015)

Para lograr que el modelo dé buenos resultados es necesario continuamente revisar y contrastar los entrenamientos con el conjunto de validación y su taza de errores, hasta dar con buenas predicciones y sin tener los problemas de *over-under-fitting* (Villa et al., 2015).

3.4.2. Entrenamiento de los algoritmos

Una vez determinado el número de épocas de entrenamiento idóneo para cada algoritmo, como lo resume la tabla 3-8, se procede a realizar cinco entrenamientos por temporada y algoritmo, con el fin de determinar si el error de entrenamiento y de validación se dispersa (la varianza tiende a ser distinta de cero). Si este fuera el caso, se continuarían con más entrenamientos para obtener una media del comportamiento del error.

Como se indicó en las pruebas de entrenamiento, se registra únicamente los valores mínimos de los errores obtenidos durante todo el entrenamiento, además se calcula y registra la varianza de estos, con el objeto de realizar una comparación de los algoritmos más adelante. Se hace uso del módulo de entrenamiento desarrollado y mostrado en la figura 18-3.

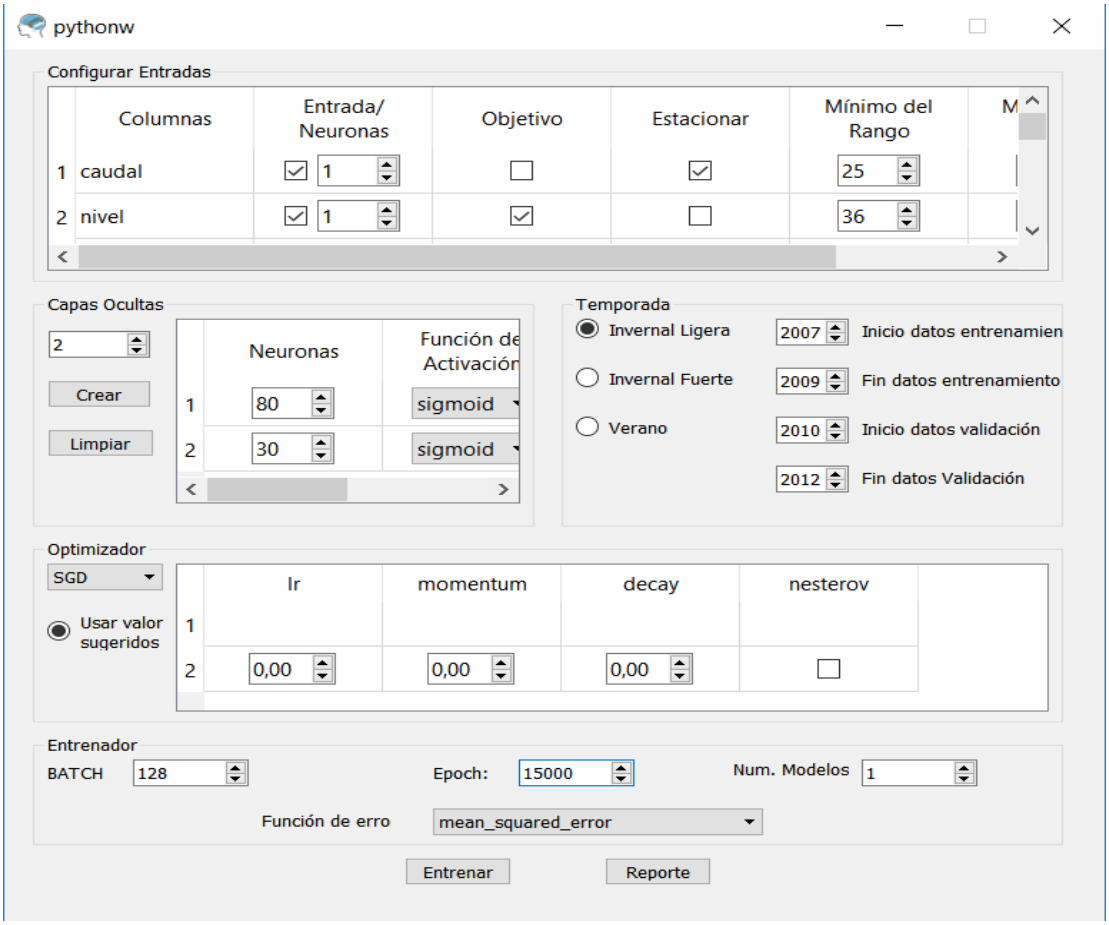


Figura 18-3: Módulo de entrenamiento.
Realizado por: Edison Chafla, 2019.

3.4.2.1. Entrenamiento para la temporada invernal fuerte

Los resultados del entrenamiento para esta temporada se presentan en la tabla 11-3, y se visualizan en el gráfico 5-3.

Tabla 11-3: Entrenamiento para la temporada invernal fuerte.

Item	**Algoritmo**	**Media Error de Entrenamiento**	**Varianza Error de Entrenamiento**	**Media Error de Validación**	**Varianza Error de Validación**
1	SGD	0,001125	4,008E-08	0,001025	6,25E-08
2	RMSprop	0,000331	1,607E-12	0,000209	2,91E-12
3	Adagrad	0,000700	4,868E-10	0,000531	5,77E-10
4	Adadelta	0,000652	1,729E-11	0,000458	4,93E-12
5	Adam	0,000341	5,184E-12	0,000230	1,09E-11
6	Adamax	0,000352	1,678E-11	0,000239	1,1E-11
7	Nadam	0,000316	8,728E-12	0,000208	2,28E-11

Realizado por: Edison Chafla, 2019.

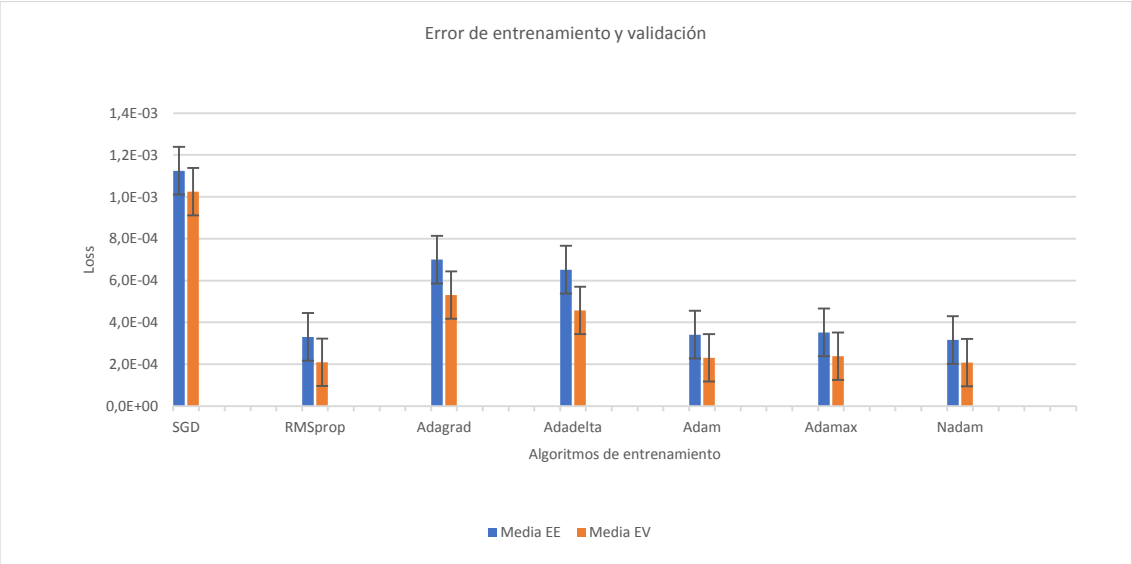


Gráfico 5-3: Error de entrenamiento y validación, temporada 1.
Realizado por: Edison Chafla, 2019.

3.4.2.2. Entrenamiento para la temporada invernal ligera

Los resultados del entrenamiento para esta temporada se presentan en la tabla 12-3, y se visualizan en el gráfico 6-3.

Tabla 12-3: Entrenamiento para la temporada invernal ligera.

Item	**Algoritmo**	**Media Error de Entrenamiento**	**Varianza Error de Entrenamiento**	**Media Error de Validación**	**Varianza Error de Validación**
1	SGD	0,000791	7,71E-10	0,000728	1,19E-09
2	RMSprop	0,00026	9,61E-12	0,000206	3,78E-12
3	Adagrad	0,000596	2,86E-10	0,00051	5,75E-10
4	Adadelta	0,000561	4,37E-10	0,000491	4,97E-10
5	Adam	0,000256	2,49E-12	0,000213	2,46E-12
6	Adamax	0,000261	2,52E-12	0,000217	2,86E-12
7	Nadam	0,000243	1,08E-12	0,000202	5,06E-12

Realizado por: Edison Chafla, 2019.

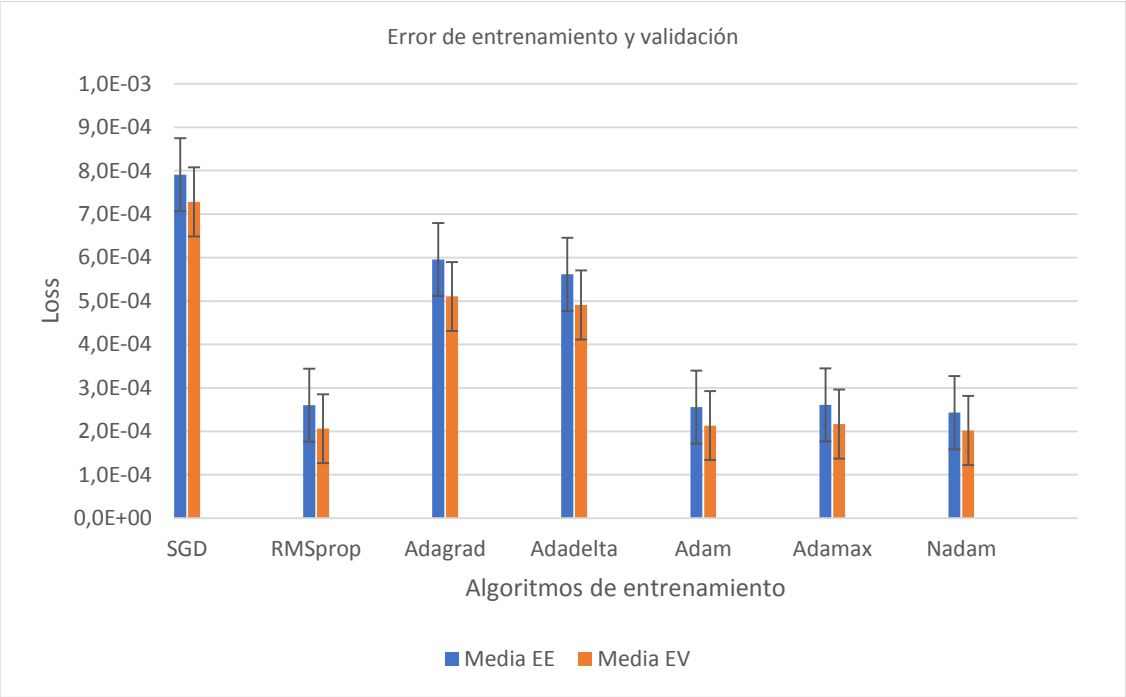


Gráfico 6-3: Error de entrenamiento y validación, temporada 2.
Realizado por: Edison Chafla, 2019.

3.4.2.3. Entrenamiento para la temporada de verano

Los resultados del entrenamiento para esta temporada se presentan en la tabla 13-3, y se visualizan en el gráfico 7-3.

Tabla 13-3: Entrenamiento para la temporada de verano.

Item	**Algoritmo**	**Media Error de Entrenamiento**	**Varianza Error de Entrenamiento**	**Media Error de Validación**	**Varianza Error de Validación**
1	SGD	0,000846	4,33E-10	0,00078	3,39E-10
2	RMSprop	0,000343	3,57E-11	0,00028	6,66E-11
3	Adagrad	0,000741	6,05E-10	0,000669	1,8E-10
4	Adadelta	0,00071	2,81E-10	0,00058	7,44E-11
5	Adam	0,00036	3,09E-11	0,0003	5,98E-11
6	Adamax	0,000362	1,27E-11	0,000298	6,66E-11
7	Nadam	0,00033	2,62E-11	0,000279	1,28E-11

Realizado por: Edison Chafla, 2019.

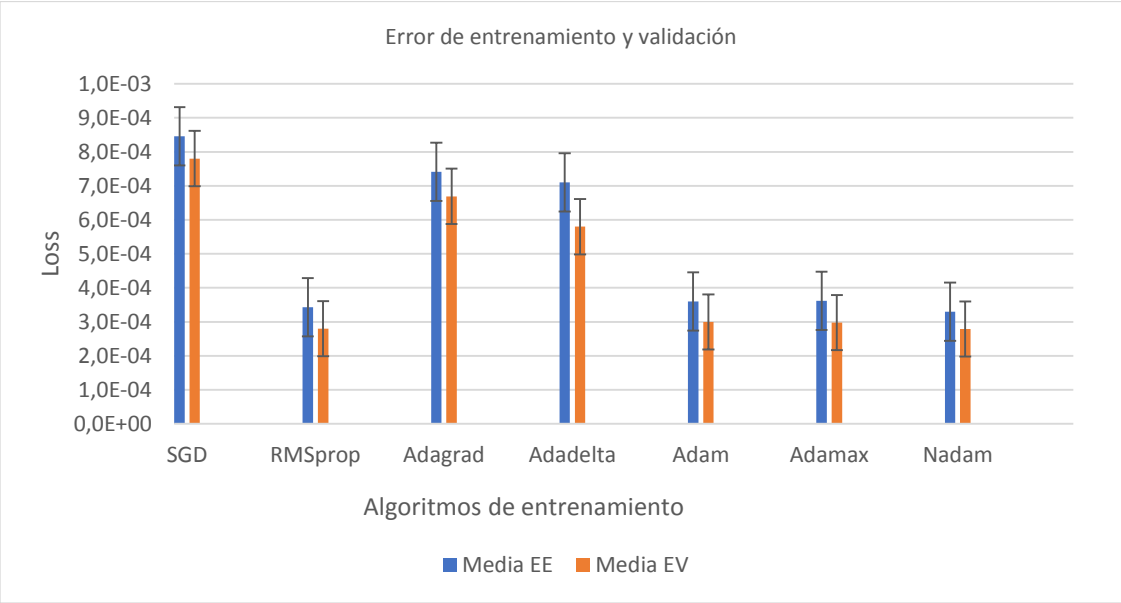


Gráfico 7-3: Error de entrenamiento y validación temporada 3.
Realizado por: Edison Chafla, 2019.

El gráfico 8-3 muestra el comportamiento del error de entrenamiento y validación en las tres temporadas, se observa que la varianza tiende a cero por lo que no es menester realizar más iteraciones de las cinco propuestas en el apartado 3.2.1.

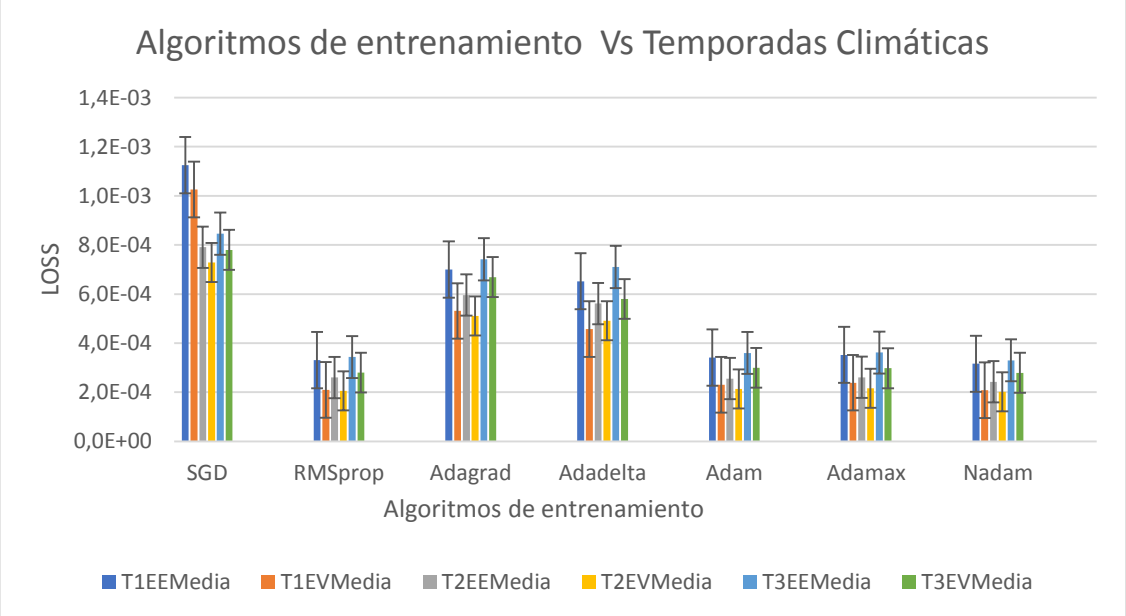


Gráfico 8-3: Entrenamiento del modelo.
Realizado por: Edison Chafla, 2019.

Adicional, de los resultados obtenidos de los entrenamientos, se consideró analizar en qué época y tiempo, los errores tienden a estabilizarse, para esto se utilizó los gráficos de los modelos en cada entrenamiento (ver Anexo 1). Los resultados se detallan en la tabla 14-3 y se visualizan en los gráficos 9-3 y 10-3.

Tabla 14-3: Estabilización de los errores, en época y tiempo.

Item	**Algoritmo**	**Temporada 1**		**Temporada 2**		**Temporada 3**	
		Época	Tiempo [min]	Época	Tiempo [min]	Época	Tiempo [min]
1	SGD	14000	88,8	12000	42,9	10000	43,8
2	RMSprop	1750	11,2	1250	5,4	1000	4,2
3	Adagrad	500	3,9	250	1,2	200	0,8
4	Adadelta	600	4,5	250	1,2	200	1,1
5	Adam	300	2,6	150	0,8	300	1,4
6	Adamax	400	3,2	200	1,3	400	2,3
7	Nadam	300	2,5	200	1,2	200	1,2

Realizado por: Edison Chafla, 2019.

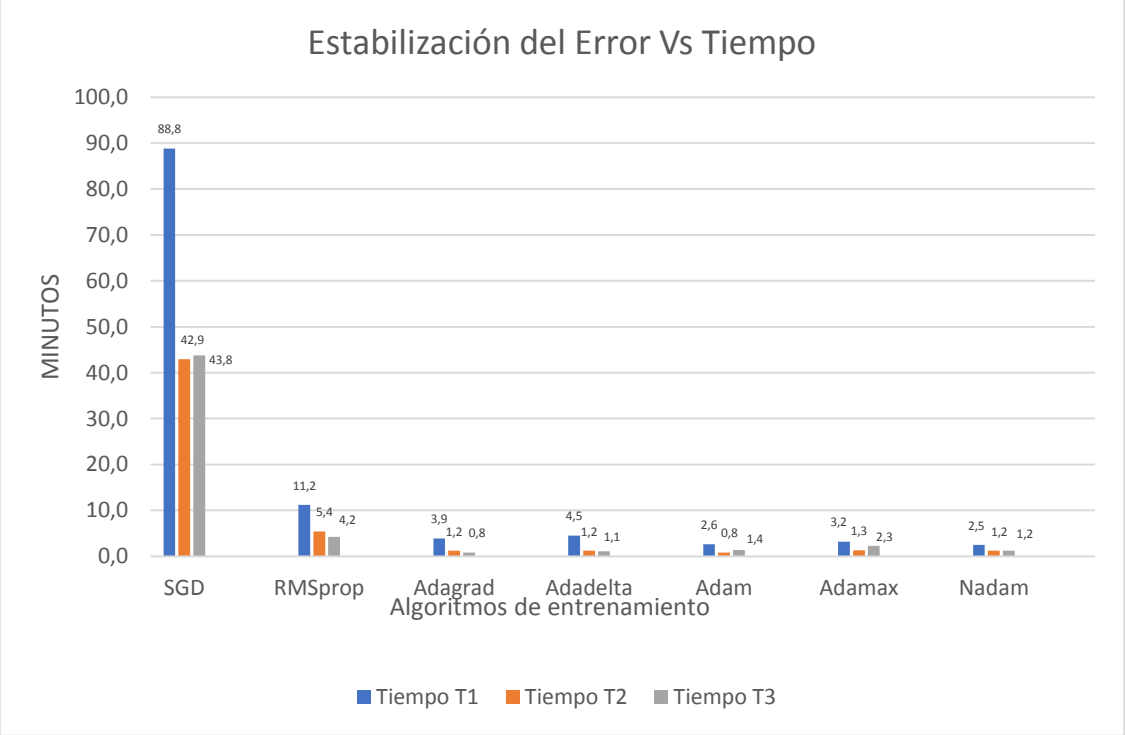


Gráfico 9-3: Estabilización del error de entrenamiento y validación en el tiempo.
Realizado por: Edison Chafla, 2019.

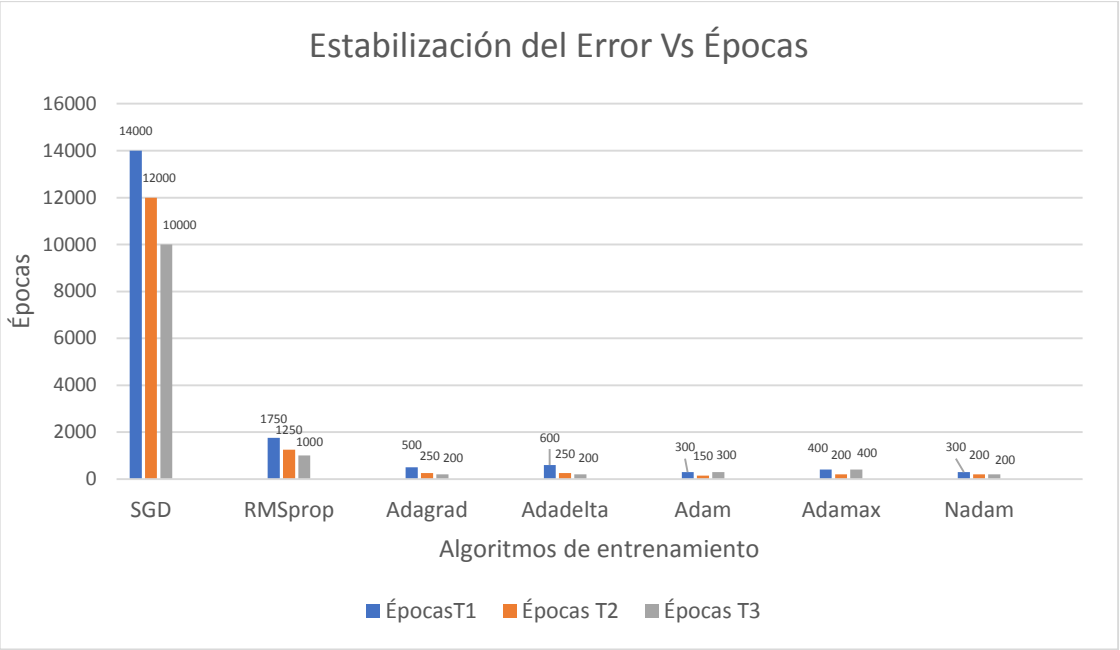


Gráfico 10-3: Estabilización del error de entrenamiento y validación en épocas.
Realizado por: Edison Chafla, 2019.

De acuerdo las pruebas iniciales de los entrenamientos, por iteración, con 15000 épocas obtuvimos un total de 24,7 horas, esto quiere decir que para 5 iteraciones nuestro tiempo total sería de 123,5 horas de entrenamiento. Gracias a las pruebas iniciales de entrenamiento y al análisis de los resultados, este tiempo se redujo a 27,4 horas, como se detalla en la tabla 15-3 y gráfico 11-3.

Tabla 15-3: Tiempo total de entrenamiento en horas.

Item	**Algoritmo**	**Temporada1 [horas]**	**Temporada 2 [horas]**	**Temporada 3 [horas]**	**Total [horas]**
1	SGD	7,93	4,47	5,48	17,87
2	RMSprop	1,06	0,72	0,70	2,49
3	Adagrad	0,48	0,29	0,25	1,02
4	Adadelta	0,62	0,39	0,45	1,47
5	Adam	0,55	0,34	0,29	1,18
6	Adamax	0,66	0,56	0,49	1,70
7	Nadam	0,69	0,48	0,49	1,67
				Total	27,4

Realizado por: Edison Chafla, 2019.

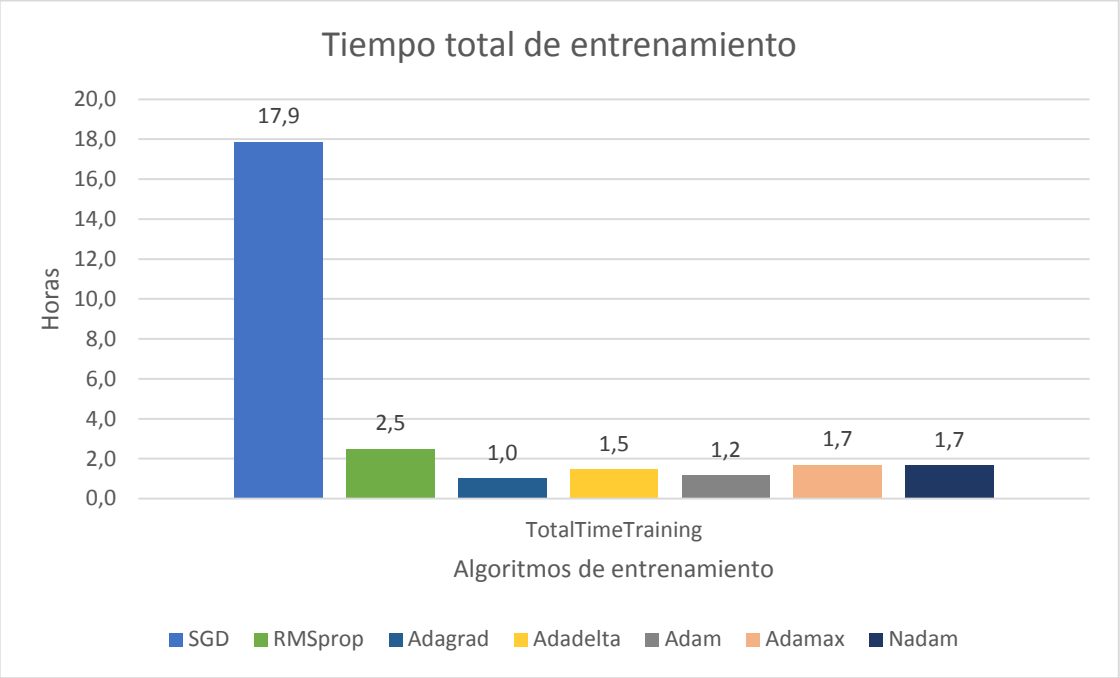


Gráfico 11-3: Tiempo total de entrenamiento.
Realizado por: Edison Chafla, 2019.

CAPÍTULO IV

4. RESULTADOS Y DISCUSIÓN

4.1. Comparación de Algoritmos de Entrenamiento

Una vez culminada la etapa de entrenamiento del modelo con cada uno de los algoritmos entrenamiento, es necesario realizar una comparación entre los mismo y establecer cuál de estos presenta un mejor rendimiento. Para ello utilizamos herramientas estadísticas que nos ayuden determinar si la diferencia en precisión de estos es significativa o no (Morales & González, 2011).

Dentro de las herramientas estadísticas utilizadas para la comparación de algoritmos de entrenamiento de RNAs esta: la evaluación de significancia con t-tes, el análisis de varianza con la prueba de Tukey y la eficiencia estadística. En este estudió se utilizó las dos últimas técnicas, debido a que la primera es utilizada para la comparación de un grupo de dos variables, en nuestro caso son 7 variables (Morales & González, 2011).

4.1.1. Análisis de la varianza y prueba Tukey.

El análisis de la varianza (ANOVA), permite estimar y evaluar una hipótesis acerca de las medias de las poblaciones (datos). Sin embargo, es preciso recordar que las hipótesis a evaluar deben formularse antes de colectar los datos y no después, en este caso se utiliza para determinar, cuando el error se mide con un conjunto de datos diferente al conjunto de datos de entrenamiento, la precisión en las medidas puede variar de la precisión verdadera y depende de los ejemplos de prueba utilizados, cabe indicar que, cuando la muestra del conjunto de datos es pequeña, la varianza es más grande (Bustos, Rodríguez, & Cantor, 2008).

La prueba de Tukey, es utilizada para probar las diferencias presentes entre las medias (μ) de tratamientos de una experiencia.

Esta técnica fue aplicada a los datos del error de entrenamiento y validación, obtenidos durante el entrenamiento de los algoritmos en las tres temporadas climáticas (Tabla 2-3), para esto se utilizó un nivel de confianza (n) del 95%.

Iniciamos formulando la siguiente pregunta:

¿Existe diferencia estadísticamente significativa en el promedio del error de entrenamiento y validación?

A partir de esta planteamos las siguientes hipótesis:

- H_0, Hipótesis nula: El promedio del error de entrenamiento (o validación) es igual en los 7 algoritmos de entrenamiento, con el 95% de confiabilidad.

$$H_0: \mu_{1=}\mu_2 = \mu_3 \dots \mu_a = \mu \quad (53)$$

- H_1, Hipótesis alterna: Al menos en un algoritmo de entrenamiento la media del error de entrenamiento (o validación) es distinta, con el 95% de confiabilidad.

$$H_0: \quad no\ es\ cierto\ \ H_0 \quad (54)$$

4.1.1.1. *Análisis de la varianza para el error de entrenamiento.*

Del resultado de la aplicación de esta técnica se obtienen la tabla 1-4 y 2-4. En la primera se presenta a modo de resumen los datos de la media y varianza y el número de muestras, es importante tener en cuenta que cada muestra, es el resultado de la media obtenida en cada entrenamiento por temporada. La segunda presenta el resultado del análisis de la varianza aplicada al error de entrenamiento.

Tabla 1-4: Análisis ANOVA del error de entrenamiento, resumen.

Algoritmos	Muestras	Suma	Media	Varianza
SGD	15	0,013803744	0,000920250	3,47044E-08
RMSprop	15	0,004672004	0,000311467	1,44549E-09
Adagrad	15	0,010184992	0,000678999	4,43077E-09
Adadelta	15	0,009617708	0,000641181	4,24178E-09
Adam	15	0,004786247	0,000319083	2,21713E-09
Adamax	15	0,004874397	0,000324960	2,21756E-09
Nadam	15	0,004445078	0,000296339	1,57765E-09

Realizado por: Edison Chafla, 2019.

Tabla 2-4: Análisis de la varianza para el error de entrenamiento.

Origen de las variaciones	Suma de cuadrados	Grados de libertad	Media de los cuadrados	F	Probabilidad	Valor crítico para F
Entre grupos	5,5345E-06	6	9,22417E-07	127,02	6,02157E-44	2,192517789
Dentro de los grupos	7,11687E-07	98	7,26212E-09			
Total	6,24619E-06	104				

Realizado por: Edison Chafla, 2019.

Debido a que el valor de la probabilidad (p) es menor que el nivel de confianza (n): $p < n$, entonces rechazamos la hipótesis nula H_0 y validamos la hipótesis alterna H_1, es decir: $6{,}02157\text{E} - 44 < 0.05$.

Lo que indica que, al menos en un algoritmo de entrenamiento la media del error de entrenamiento (o validación) es distinta, con el 95% de confiabilidad. Con estos datos obtenidos del ANOVA procedemos a realizar la prueba de Tukey, para encontrar las diferencias significativas entre los algoritmos de entrenamiento. Para esto es necesario calcular la *HDS(diferencia honestamente significativa)*.

$$HDS = VC\sqrt{\frac{MSe}{N}} \qquad (55)$$

Donde:

VC: es el valor crítico (*VC)*, para los 98 grados de libertad, este valor de acuerdo con la Tabla de Tukey es: $VC = 4.24$

MSe: Error cuadrático medio:

$$MSe = \frac{Suma\ de\ Cuadrados}{Grados\ de\ Libertad} \qquad (56)$$

$$MSe = \frac{7.11687E-07}{98}$$

$$MSe = 7.2621E - 09$$

N: Número de elementos en los grupos $N = 15$

Con estos datos obtenemos: $HDS = 9{,}32936\text{E} - 05$

El valor del HDS es comparado con la media (μ) de cada algoritmo de entrenamiento, para determinar la relación que existe entre cada uno de los algoritmos de entrenamiento, para esto restamos el valor de la media (μ) de cada algoritmo con respecto al otro, si este valor es superior al *HDS*, entonces existe diferencia entre ambos, si *HDS* es menos no existe diferencia, es decir:

$$HDS < \mu_1 - u_2 \; ; \; existe\; diferencia \qquad (57)$$

$$HDS < \mu_1 - u_2 \; ; \; No\; existe\; diferencia \qquad (58)$$

En la tabla 3-4, se establecieron estos resultados, en color azul se encuentran los algoritmos donde no existe diferencia y en color tomate los algoritmos en donde existe diferencia.

Tabla 3-4: Comparativo de los errores de entrenamiento de cada algoritmo.

	SGD	RMSprop	Adagrad	Adadelta	Adam	Adamax	Nadam
SGD		6,088E-04	2,413E-04	2,791E-04	6,012E-04	5,953E-04	6,239E-04
RMSprop			-3,675E-04	-3,297E-04	-7,616E-06	-1,349E-05	1,513E-05
Adagrad				3,782E-05	3,599E-04	3,540E-04	3,827E-04
Adadelta					3,221E-04	3,162E-04	3,448E-04
Adam						-5,877E-06	2,274E-05
Adamax							2,862E-05
Nadam							

Realizado por: Edison Chafla, 2019.

4.1.1.2. *Análisis de la varianza para el error de validación.*

Se contempla el mismo método de análisis que en el error de entrenamiento, y se obtienen los siguientes resultados:

Tabla 4-4: Análisis ANOVA del error de validación, resumen.

Algoritmos	Muestras	Suma	Media	Varianza
SGD	15	0,012666622	0,000844441	3,63129E-08
RMSprop	15	0,003477225	0,000231815	1,26937E-09
Adagrad	15	0,008552948	0,000570197	5,69953E-09
Adadelta	15	0,007643419	0,000509561	3,01269E-09
Adam	15	0,003716301	0,000247753	1,51413E-09
Adamax	15	0,003768466	0,000251231	1,27654E-09
Nadam	15	0,003445952	0,00022973	1,32589E-09

Realizado por: Edison Chafla, 2019.

Tabla 5-4: Análisis de la varianza para el error de entrenamiento.

Origen de las variaciones	**Suma de cuadrados**	**Grados de Libertad**	**Media de los cuadrados**	**F**	**Probabilidad**	**Valor crítico para F**
Entre grupos	5,10094E-06	6	8,50156E-07	118,05	1,40242E-42	2,192517789
Dentro de los grupos	7,05754E-07	98	7,20158E-09			
Total	5,80669E-06	104				

Realizado por: Edison Chafla, 2019.

Debido a que el valor de la: $Probabilidad < n$, entonces rechazamos la hipótesis nula H_0 y validamos la hipótesis alterna H_1, es decir:

$$1.4024\text{E} - 42 < 0.05$$

Lo que indica que, al menos en un algoritmo de entrenamiento la media del error de entrenamiento (o validación) es distinta, con el 95% de confiabilidad. Con estos datos obtenidos del ANOVA procedemos a realizar la prueba de Tukey, para encontrar las diferencias significativas entre los algoritmos de entrenamiento. Para esto es necesario calcular la *HDS(diferencia honestamente significativa)*.

$$HDS = VC\sqrt{\frac{MSe}{N}} \qquad (59)$$

Obtenemos: $$HDS = 9{,}2904\text{E} - 05$$

El valor del HDS es comparado con la media (μ) de cada algoritmo de entrenamiento, para determinar la relación que existe entre cada uno de los algoritmos de entrenamiento, para esto restamos el valor de la media (μ) de cada algoritmo con respecto al otro, si este valor es superior al *HDS*, entonces existe diferencia entre ambos, si *HDS* es menos no existe diferencia., es decir:

$$HDS < \mu_1 - u_2 \; ; \; existe\; diferencia$$

$$HDS < \mu_1 - u_2 \; ; \; No\; existe\; diferencia$$

En la Tabla 6-4, se establecieron estos resultados, en color azul se encuentran los algoritmos donde no existe diferencia y en color tomate los algoritmos en donde existe diferencia.

Tabla 6-4: Comparativo de los errores de validación con cada algoritmo.

	SGD	RMSprop	Adagrad	Adadelta	Adam	Adamax	Nadam
SGD		6,126E-04	2,742E-04	3,349E-04	5,967E-04	5,932E-04	6,147E-04
RMSprop			-3,384E-04	-2,777E-04	-1,594E-05	-1,942E-05	2,085E-06
Adagrad				6,064E-05	3,224E-04	3,190E-04	3,405E-04
Adadelta					2,618E-04	2,583E-04	2,798E-04
Adam						-3,478E-06	1,802E-05
Adamax							2,150E-05
Nadam							

Realizado por: Edison Chafla, 2019.

Los valores de la varianza tanto para el error de entrenamiento y el error de validación son representados en el gráfico 1-4.

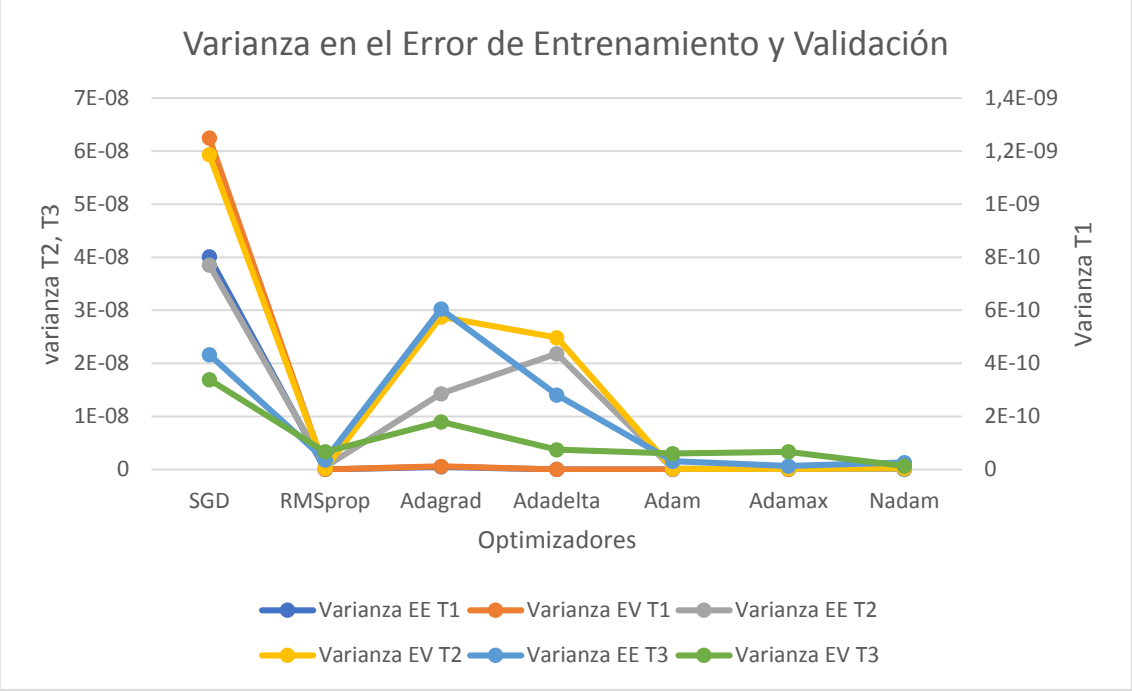


Gráfico 1-4: Varianza en el error de entrenamiento y validación.
Realizado por: Edison Chafla, 2019.

4.1.2. Eficiencia estadística

La eficiencia estadística indica que un estimador es más eficiente o más preciso que otro, el de menor varianza se llama estimador eficiente, mientras el de mayor varianza será un estimador ineficiente, por ejemplo, si θ_1 y θ_2 ambos son estimadores de θ, entonces (Guillen, 2012):

$$Var(\theta_1) < Var(\theta_2)$$

Se indica que θ_1 es mas eficiente que θ_2, por lo tanto, se concluye que un estimador es más eficiente (preciso), cuando menor es su varianza (Guillen, 2012). Las varianzas obtenidas del error de entrenamiento y el error de validación en las tres temporadas climáticas (Tabla 2-3) del algoritmo Nadam y SGD se detallan en la tabla 7-4.

Tabla 7-4: Análisis de la varianza para el error de entrenamiento y validación.

Algoritmos	Varianza EE T1	Varianza EV T1	Varianza EE T2	Varianza EV T2	Varianza EE T3	Varianza EV T3
SGD	4,00841E-08	6,25012E-08	7,71E-10	1,19E-09	4,33E-10	3,39E-10
Nadam	8,72842E-12	2,28496E-11	1,08E-12	5,06E-12	2,62E-11	1,28E-11

Realizado por: Edison Chafla, 2019.

De acuerdo con el gráfico 1-4 y tabla 7-4, se determina que el algoritmo Nadam presenta una varianza menor en las tres temporadas climáticas tanto para el error de entrenamiento y validación, en comparación con el algoritmo SGD.

$$Var(Nadam) < Var(SGD) \quad (60)$$

Basado en lo anterior se determina que:

- El algoritmo Nadam, es el más eficiente.
- El Algoritmo SGD, es ineficiente.

4.2. Validación de los modelos

Finalmente se procede a realizar la evaluación de los modelos determinados como más y menos eficientes (precisos), en el módulo de predicción de nivel de agua de embalse de la plataforma de software, como se observa en la figura 1-4. Este proceso se realizó bajo los siguientes parámetros:

- Algoritmo Nadam y SGD.
- Conjunto de evaluación del año 2014, distinto del conjunto de entrenamiento y validación.
- Tres temporadas climáticas.
- Validación con muestras de 10 días.
- Umbrales de 4, 8, 24 y 48 horas.

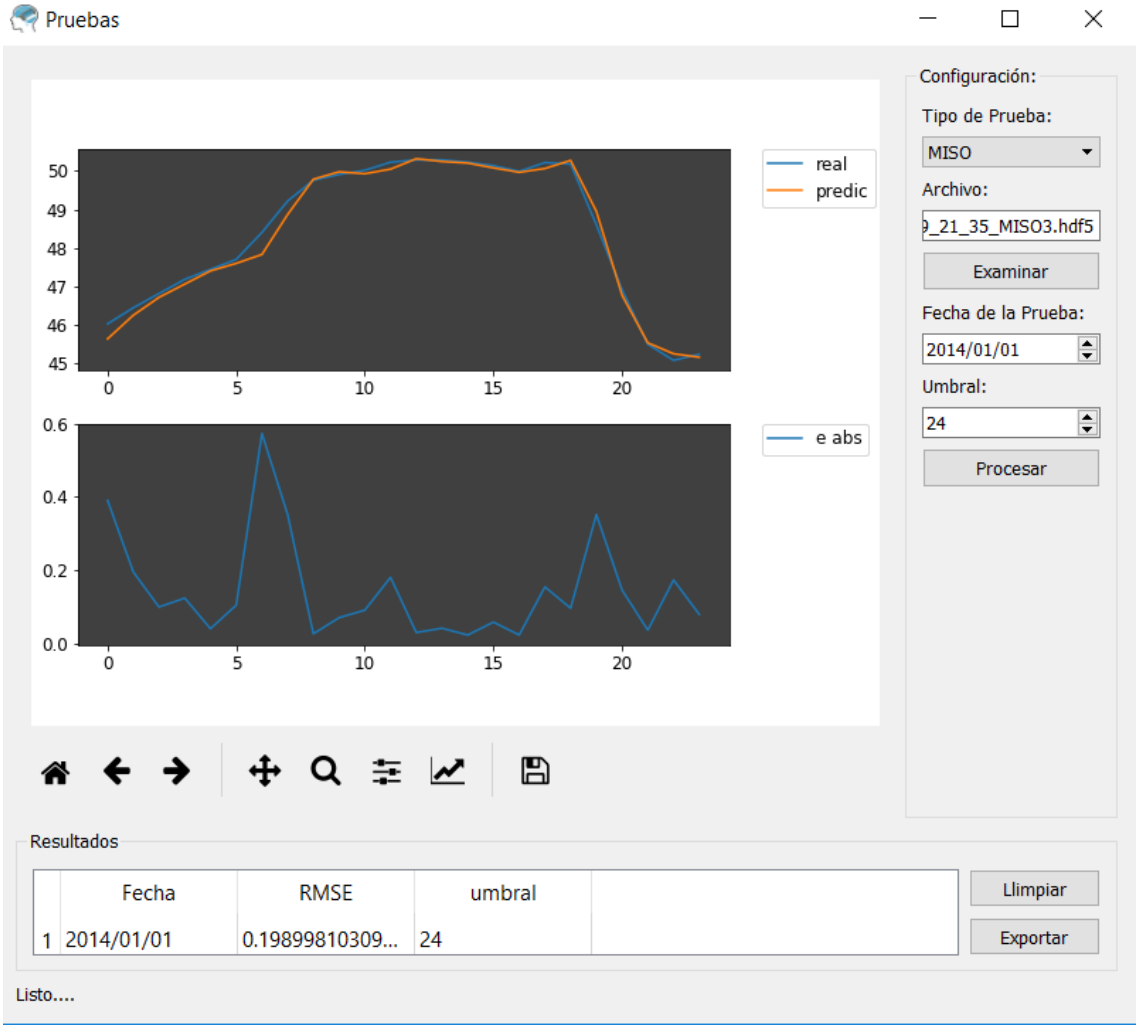


Figura 1-4: Evaluación de los modelos entrenados.
Realizado por: Edison Chafla, 2019.

En el módulo de pruebas de la plataforma una vez seleccionados los parámetros establecidos para el entrenamiento, genera un archivo en formato *.xlsx, en el cual incluye el valor del RMSE (error cuadrático medio) de la variable nivel. Una vez realizadas las pruebas se calculó el promedio de RMSE de los 10 días seleccionados para cada análisis, se obtiene los siguientes resultados detallados en la tabla 8-4.

Tabla 8-4: RMSE promedio del modelo de la presa.

Algoritmo	**Temporada**	**Umbral Horas**	**RMSE Promedio [m]**	**Min RMSE [m]**	**Max RMSE [m]**
SGD	Invernal Fuerte	4	0,391647	0,135600	0,553381
		8	0,392976	0,201337	0,606749
		24	0,407186	0,337605	0,509727
		48	0,403287	0,338025	0,465412
	Invernal Ligera	4	0,357179	0,090600	0,869124
		8	0,386650	0,204000	0,658610
		24	0,374738	0,222991	0,509068
		48	0,406742	0,306073	0,508519
	Verano	4	0,345530	0,100736	0,524029
		8	0,378624	0,185296	0,540933
		24	0,490837	0,313222	0,723011
		48	0,513799	0,336963	0,707330
Nadam	Invernal Fuerte	4	0,108883	0,035876	0,204691
		8	0,121077	0,052596	0,207652
		24	0,110025	0,058382	0,224007
		48	0,112726	0,065765	0,195028
	Invernal Ligera	4	0,190445	0,041521	0,322500
		8	0,182604	0,061096	0,344913
		24	0,159224	0,096815	0,216750
		48	0,163755	0,118019	0,207596
	Verano	4	0,131787	0,064824	0,232400
		8	0,185389	0,069992	0,311182
		24	0,176891	0,093733	0,236966
		48	0,182319	0,095510	0,218915

Realizado por: Edison Chafla, 2019.

Se utilizaron los umbrales de 4, 8, 24, 48 horas, tomados del estudio de investigación de G. Asqui, en donde identifica estos umbrales basados en los picos del PSD de la señal de nivel.

Estos resultados son visualizados en los gráficos 2-4 y 3-4, que representa la evolución del RMSE del nivel de agua de embalse de la presa, cada temporada inicia con un umbral de 4 horas y termina en 48 horas. De acuerdo con esto podemos verificar y confirmar lo analizado en el apartado de la comparación de los algoritmos, en el cual identificamos al algoritmo Nadam como el más eficiente, existe un rendimiento superior de este en comparación al algoritmo de entrenamiento SGD.

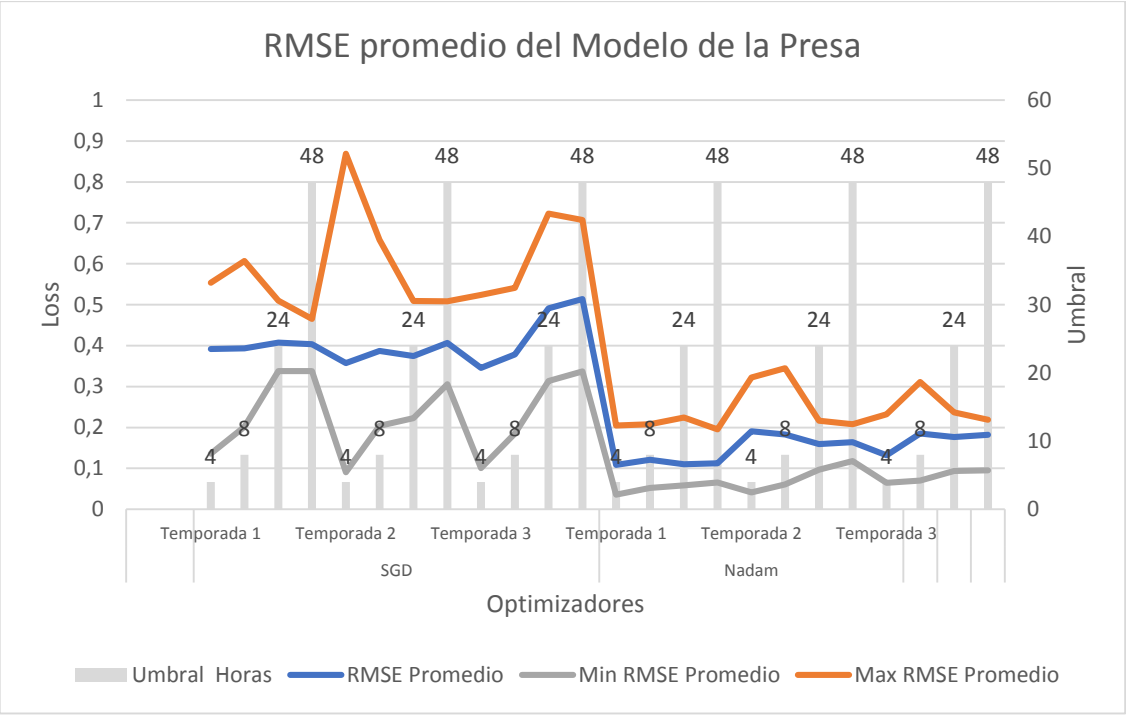


Gráfico 2-4: Evaluación del algoritmo SGD y Nadam.
Realizado por: Edison Chafla, 2019.

CONCLUSIONES

A continuación, se presenta las conclusiones obtenidas durante el desarrollo de esta tesis, la cual expone el análisis del rendimiento de algoritmos de entrenamiento de redes neuronales artificiales, aplicadas al modelamiento dinámico de represas hidroeléctricas, mediante el error de predicción del nivel de embalse de agua.

- La evaluación del rendimiento de los algoritmos de entrenamiento de RNAs de la herramienta Open Source Keras, en función del tiempo de entrenamiento, error de entrenamiento y error de validación en la predicción del nivel de embalse de agua de la represa hidroeléctrica Agoyán, permitió identificar que el tipo de algoritmo de entrenamiento si influye en el error de nivel de embalse de agua ya que se identificaron diferencias significativas entre los algoritmos SGD y Nadam (véase Tabla 3-4 y 6-4), y determinó que el algoritmo SGD, es el menos eficiente y el algoritmo Nadam es el más eficiente del grupo, esto se confirmó al utilizar sus modelos en el módulo de pruebas del error de nivel de agua de embalse de la presa (véase Tabla 8-4).

- El estudio del arte de los algoritmos de entrenamiento de la librería Keras permitió identificar los algoritmos existentes para el desarrollo de la plataforma de software para el análisis del error de entrenamiento y validación de dichos algoritmos.

- Las pruebas iniciales previo al entrenamiento de los algoritmos de entrenamiento basados en el tiempo de entrenamiento, error de entrenamiento y error de validación, permitieron reducir los tiempos de entrenamiento consiguiendo una disminución en 96,1 hora de entrenamiento. Además, de ello conseguir que los modelos no sufran un sobreajuste (véase Tabla 9-3 y 15-3).

- No se identificaron diferencias significativas entre los algoritmos Nadam, Adam y Adamax, por lo cual se utilizó la eficiencia estadística para determinar cuál es más y menos eficiente. Se determinó que el algoritmo Nadan es el más eficiente de este grupo.

- Se alcanzó un predictor de nivel efectivo hasta un umbral de 48 horas del modelo entrenado con el algoritmo Nadam, consiguiendo un RMSE promedio de 0,112726 [m], 0,163755[m] y 0,182319 [m] para las temporadas 1, 2 y 3 respectivamente (véase Tabla 9.4).

RECOMENDACIONES

- De acuerdo con la cantidad de datos que se utilice para los entrenamientos de modelo de las RNAs se debe dimensionar las características de la GPU, lo que permitirá ahorros en tiempos de entrenamiento.

- Para este caso de estudio se utilizó el backend de TensorFlow, pero existe actualmente otros backends como: Theano y CNTK, se podría entrenar los algoritmos en estas estructuras y probar su rendimiento.

- Para el desarrollo de una plataforma de software es recomendable la utilización de multithreading en su diseño y programación ya que divide las aplicaciones en módulos funcionales bajo un mismo direccionamiento virtual permitiendo acelerar el rendimiento del procesamiento del sistema.

BIBLIOGRAFÍA

Asqui, G. (2017). *Predicción del nivel de agua del embalse, basado en redes neuronales, para la mejora de la planificación de producción de energía en la Central Hidroeléctrica Agoyán.*(tesis de maestría), Escuela Superiro Politécnica de Chimborazo, Ecuador.

Asqui, G., & Hernánez, J. (2017). Multistep-ahead Streamflow and Reservoir Level Prediction Using ANNs for Production Planning in Hydroelectric Stations. *EEE International Conference on Machine Learning and Applications*, 479–484. https://doi.org/10.1109/ICMLA.2017.0-115

Azagra, J. (2017). *Control robusto cuantitativo de sistemas con múltiples entradas de actuación y una salida objeto de control*. (tesis doctoral), Universidad de la Rioja, España.

BANCO MUNDIAL. (2017). Energía www.bancomundial.org. Retrieved from https://www.bancomundial.org/es/news/feature/2018/05/18/sustainable-development-goal-7-energy-access-all

Bertona, L. (2005). *Entrenamiento de redes neuronales basado en algoritmos evolutivos*. (tesis de pregrado), Universidad de Buenos Aires, Argentina.

Bustos, A., Rodríguez, D., & Cantor, F. (2008). Lo que los biólogos pueden usar para analizar sus datos experimentales. *Revista, Faculta de Ciencias Básicas, Universidad Militar de "Nueva Granada," 4*(February 2017), 143–148.

Ceballos, J. (2017). Curso Python Hilos (Threading). Retrieved June 27, 2018, from https://www.redeszone.net/2017/07/13/curso-python-volumen-xx-hilos-parte-i/

Chang, F., & Chang, Y. (2006). Adaptive neuro-fuzzy inference system for prediction of water level in reservoir. *Advances in Water Resources*, *29*(1), 1–10. https://doi.org/10.1016/J.ADVWATRES.2005.04.015

Chollet, F. (2007). *Deep Learning with Python* (Vol. 80). https://doi.org/citeulike-article-id:10054678

Chollet, F. (2015a). Keras Documentation. Retrieved June 18, 2018, from https://keras.io/

Chollet, F. (2015b). Optimizers - Keras Documentation. Retrieved December 4, 2017, from https://keras.io/optimizers/

CONELEC. Plan Maestro de Electrificación 2013-2022, 1 Plan Maestro de Electrificación 2013-2022 116 (2013). https://doi.org/10.1017/CBO9781107415324.004

Correa, J., Morales, E., Huerta, J., González, J., & Cárdenas, C. (2016). Sistema de adquisición de señales SEMG para la detección de fatiga muscular. *Revista Mexicana de Ingenieria Biomedica*, *37*(1), 17–27. https://doi.org/10.17488/RMIB.37.1.4

Correa, R. Decreto 1014 (2010).

Cortés, C. (2017). *Herramientas Modernas En Redes Neuronales: La Librería Keras*. https://doi.org/1617 032 CO

Dozat, T. (2016). Incorporating Nesterov Momentum into Adam. *ICLR Workshop*, (1), 2013–2016.

Galán, H., & Martínez, A. (2010). Inteligencia artificial . Redes neuronales y Aplicaciones. *Universidad Carlos III de Madrid, Journal*, 8. Retrieved from http://www.it.uc3m.es/jvillena/irc/practicas/10-11/06mem.pdf

Galarza, L. (2017). Estadística Anual y Multianual del Sector Eléctrico Ecuatoriano, 208. Retrieved from http://www.regulacionelectrica.gob.ec/wp-content/uploads/downloads/2017/08/Estadística-anual-y-multianual-sector-eléctrico-2016.pdf

Ghiassi, M., Saidane, H., & Zimbra, D. K. (2005). A dynamic artificial neural network model for forecasting time series events. *International Journal of Forecasting*, *21*(2), 341–362. https://doi.org/10.1016/J.IJFORECAST.2004.10.008

Ghorbani, M., Khatibi, R., & Aytek, A. (2010). Sea water level forecasting using genetic programming and comparing the performance with Artificial Neural Networks. *Computers & Geosciences*, *36*(5), 620–627. https://doi.org/10.1016/J.CAGEO.2009.09.014

Guillen, A. (2012). Estimaciones Estadísticas : Un Acercamiento Analítico. *International Journal of Good Conscience*, *5*(1), 237–255.

INEC. (2017). Proyecciones Poblacionales.

Isasi, P., & Galván León, I. (2004). *Redes Neuronales Artificiales Un Enfoque Practico*. Madrid, 2004.

Morales, E., & González, J. (2011). *Aprendizaje Computacional*. *Instituto Nacional de Astrofísica, Óptica y Electrónica*. Retrieved from http://ccc.inaoep.mx/~jagonzalez/ML/principal.pdf

Raman, H., & Sunilkumar, N. (1995). Multivariate modelling of water resources time series using artificial neural networks. *Hydrological Sciences Journal*, *40*(2), 145–163. https://doi.org/10.1080/02626669509491401

REN21. (2017). *Renewables 2017: global status report*. https://doi.org/10.1016/j.rser.2016.09.082

Ruder, S. (2016). An overview of gradient descent optimization algorithms, 1–14. https://doi.org/10.1111/j.0006-341X.1999.00591.x

SENPLADES. (2013). *Plan Nacional del Buen Vivir Ecuador 2013-2017* (Vol. 1). Retrieved from https://www.unicef.org/ecuador/Plan_Nacional_Buen_Vivir_2013-2017.pdf

Taormina, R., Chau, K., & Sethi, R. (2012). Artificial neural network simulation of hourly groundwater levels in a coastal aquifer system of the Venice lagoon. *Engineering Applications of Artificial Intelligence*, *25*(8), 1670–1676. https://doi.org/10.1016/J.ENGAPPAI.2012.02.009

Torres, J. (2017). Introducción práctica al Deep Learning con TensorFlow de Google: Supercomputing for Artificial Intelligence and Deep Learning. Retrieved June 18, 2018, from http://jorditorres.org/introduccion-practica-al-deep-learning-con-tensorflow-de-google-parte-5/

Villa, F., Velásquez, J., & Sánchez, P. (2015). Control del sobreajuste en redes neuronales tipo cascada correlación aplicado a la predicción de precios de contratos de electricidad. *Revista Ingenierías Universidad de Medellín Control*, *14*(26), 161–176.

Zaldivar, J., Gutiérrez, E., & Galván, I. (2000). Forecasting high waters at Venice Lagoon using chaotic time series analysis and nonlinear neural networks. *Journal of Hydroinformatics*, *2*(2000), 61–84. Retrieved from http://publications.jrc.ec.europa.eu/repository/handle/JRC18675

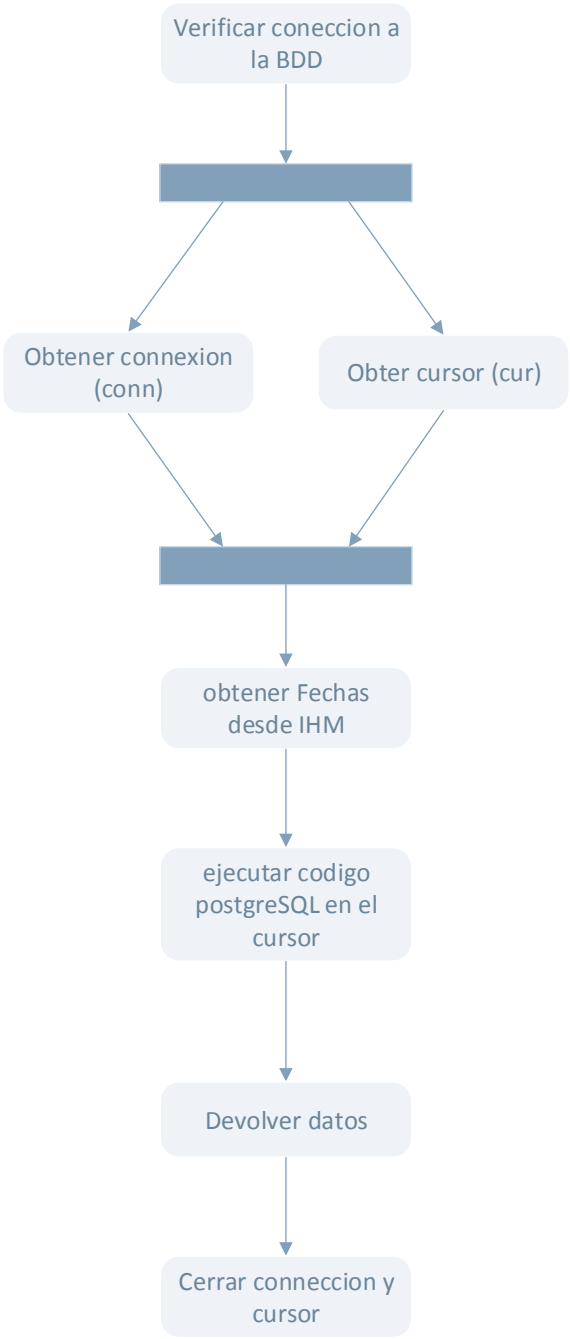
Zeiler, M. (2012). ADADELTA: An Adaptive Learning Rate Method. *Cornell University, Journal*. Retrieved from http://arxiv.org/abs/1212.5701

Zhang, G., Patuwo, B., & Hu, M. (2001). A simulation study of artificial neural networks for nonlinear time-series forecasting. *Computers & Operations Research*, *28*(4), 381–396. https://doi.org/10.1016/S0305-0548(99)00123-9

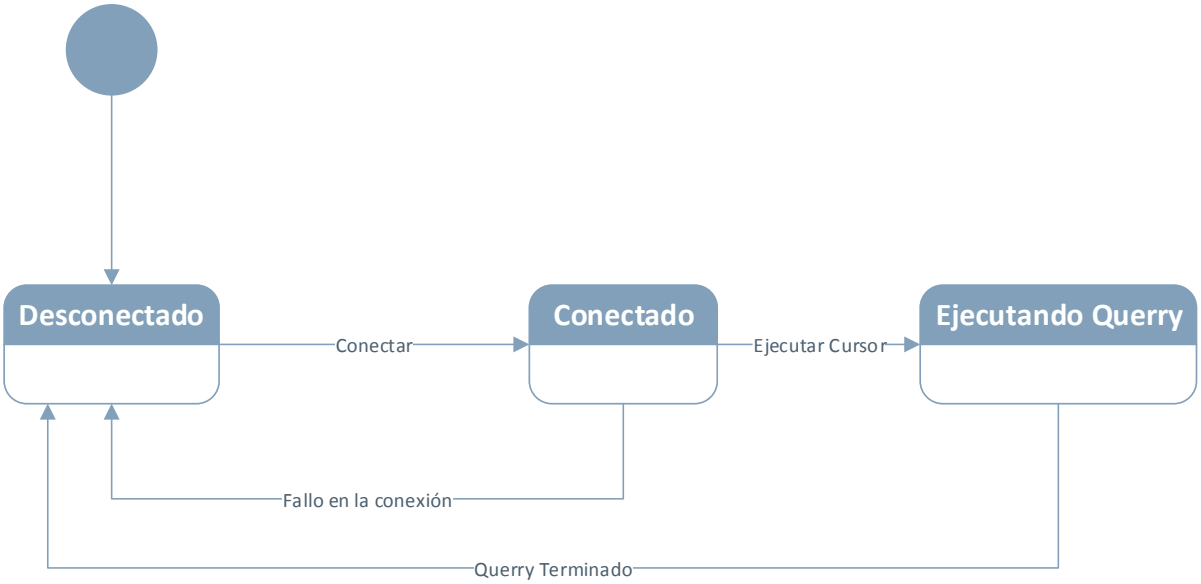
Zuñiga, A., & Jórdan, C. (2005). Pronóstico de caudales medios mensuales empleando Sistemas Neurofuzzy. *Media*, *18*, 17–23.

ANEXOS

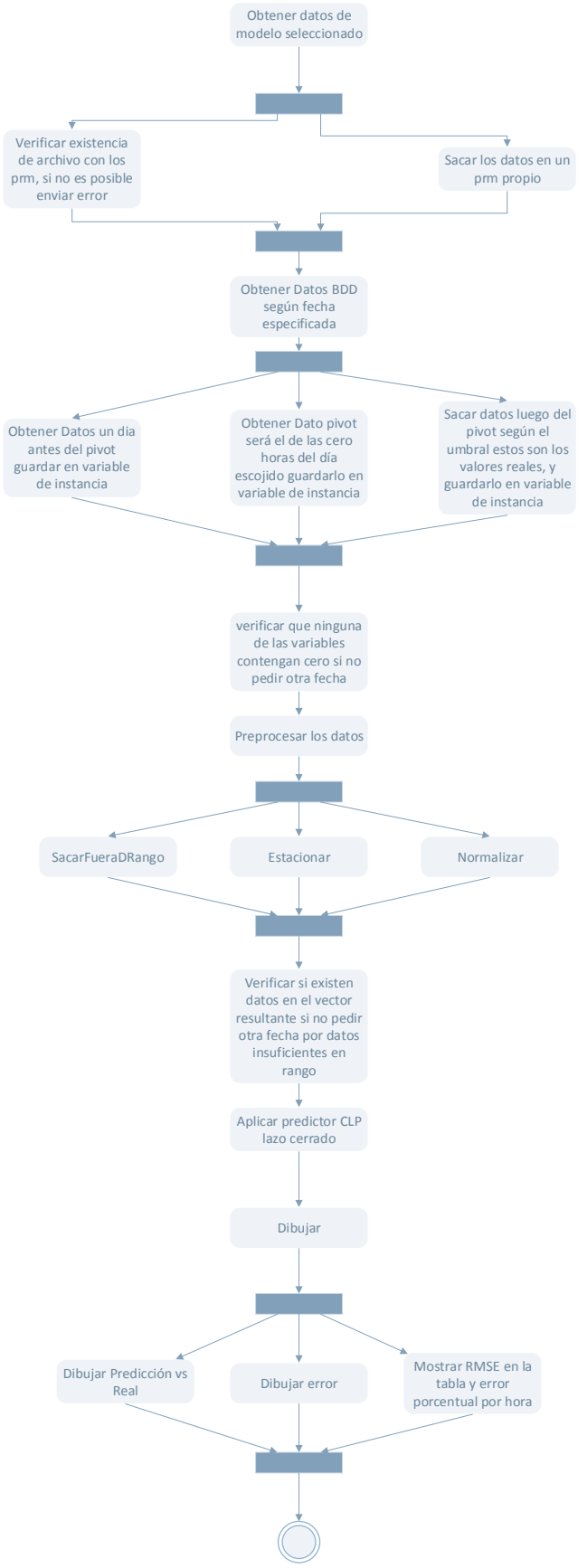
ANEXO A: Acceso a la base de datos.



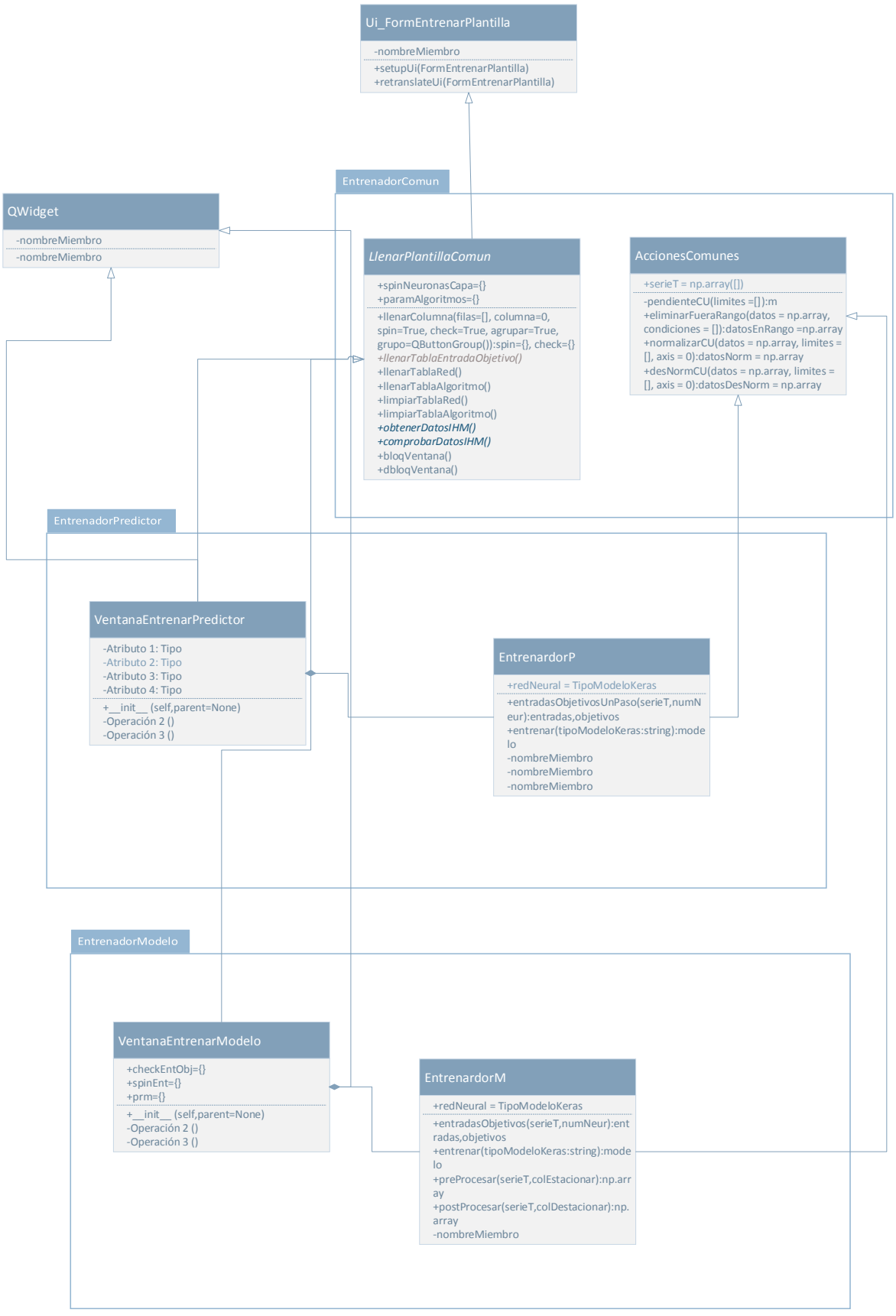
ANEXO B: Estados de la base de datos



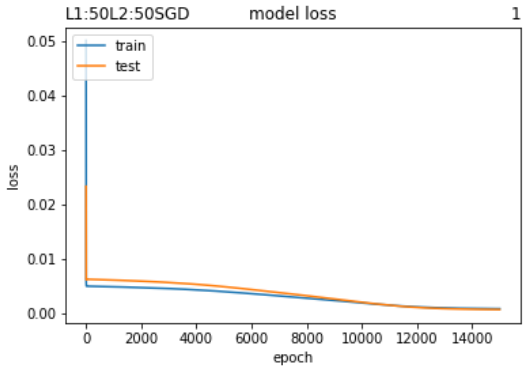
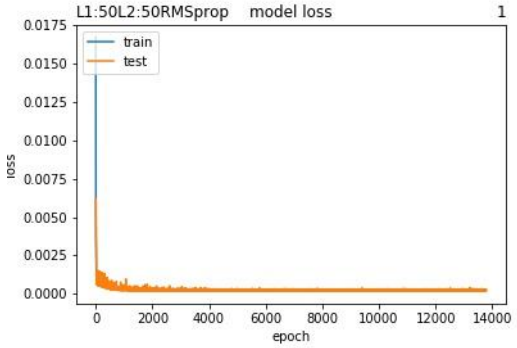
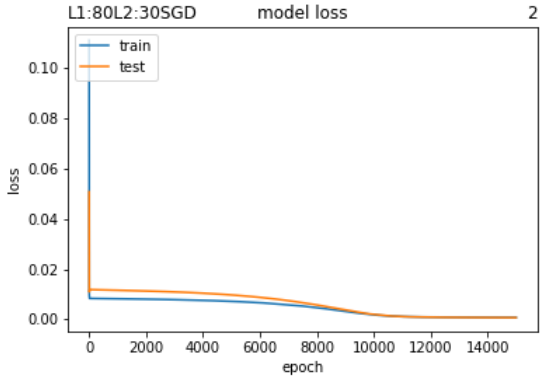
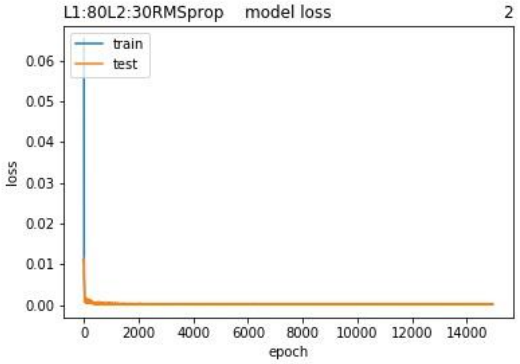
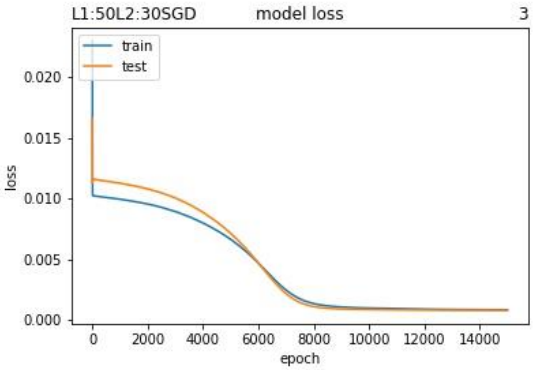
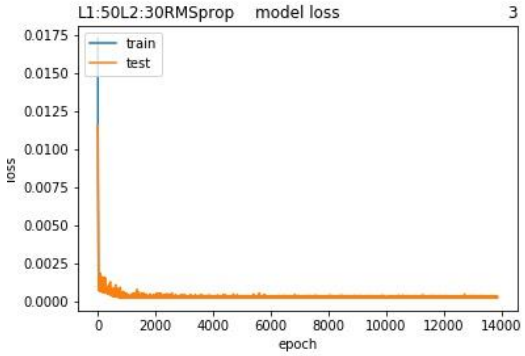
ANEXO C: Actividades de prueba de la RNA.



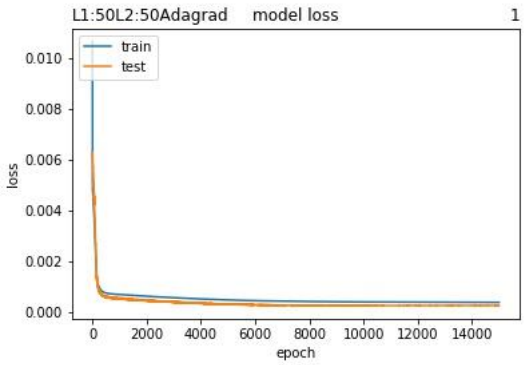
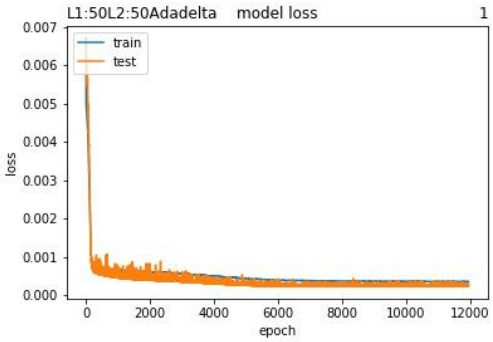
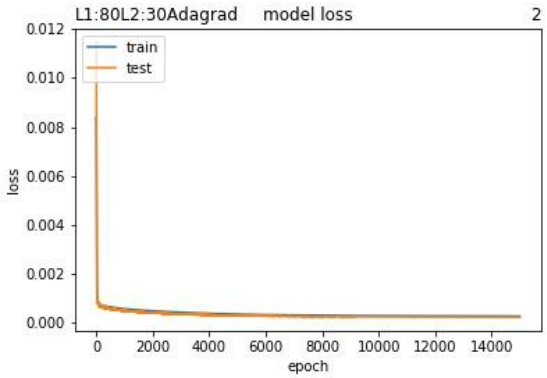
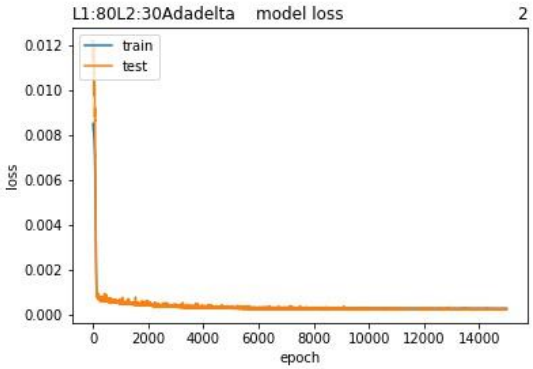
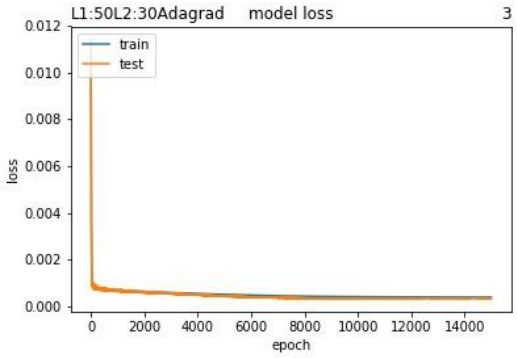
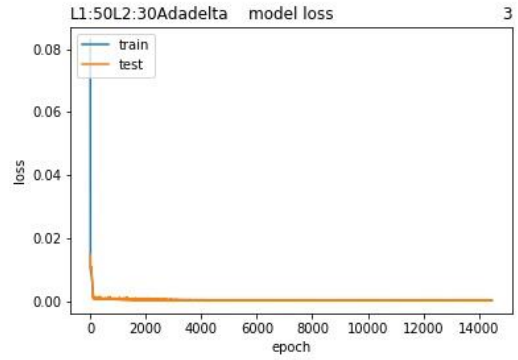
ANEXO D: Vista estática de la ventana entrenador predictor.



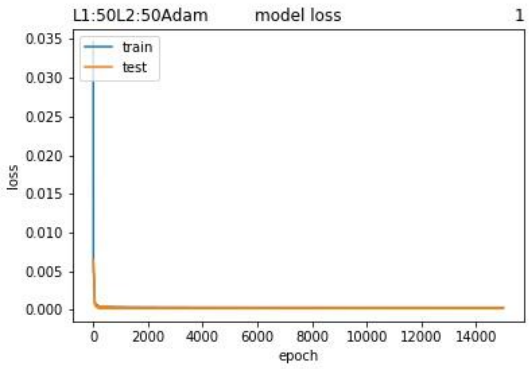
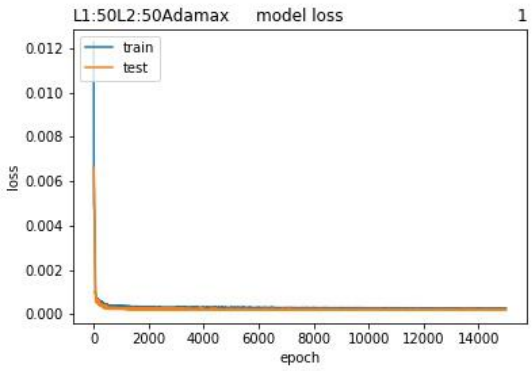
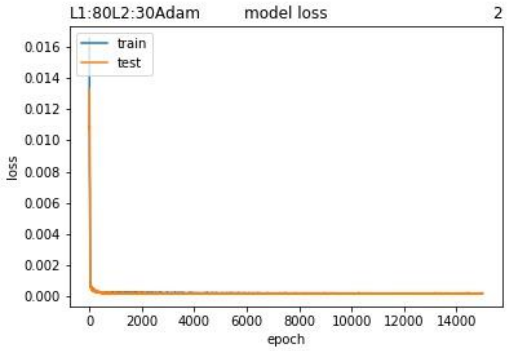
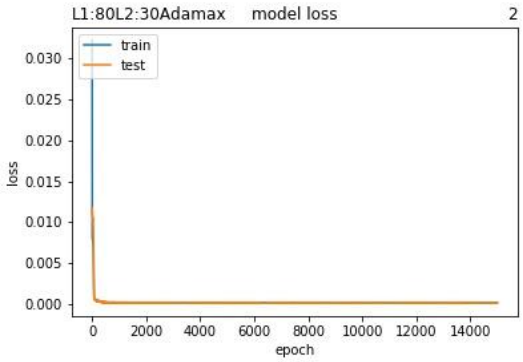
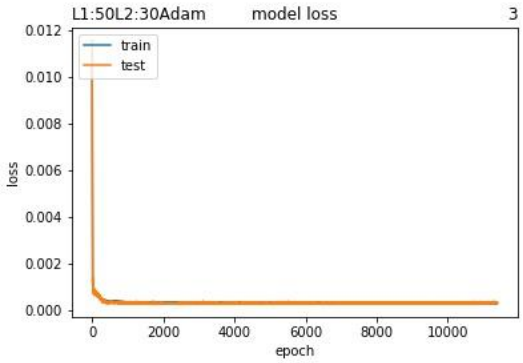
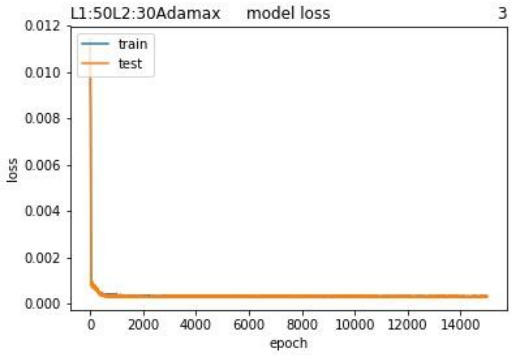
ANEXO E: Validación de parámetros para el entrenamiento con 15000 épocas.

1. SGD	2. RMSprop
	
	
	

SGD	Tiempo [min]	Épocas	RMSprop	Tiempo [min]	Épocas
Temporada 1	88,48	15000	Temporada 1	64,04	2000
Temporada 2	55,68		Temporada 2	53,81	
Temporada 3	62,68		Temporada 3	73,26	

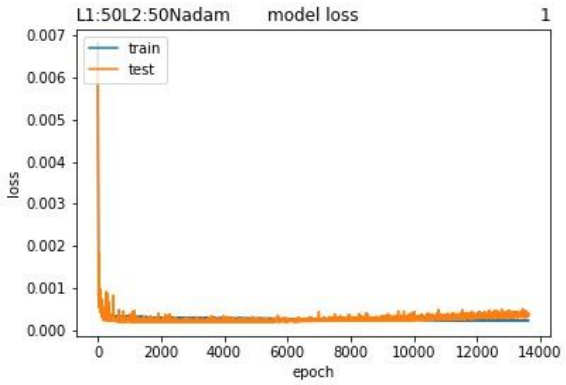
3. Adagrad	4. Adadelta
	
	
	

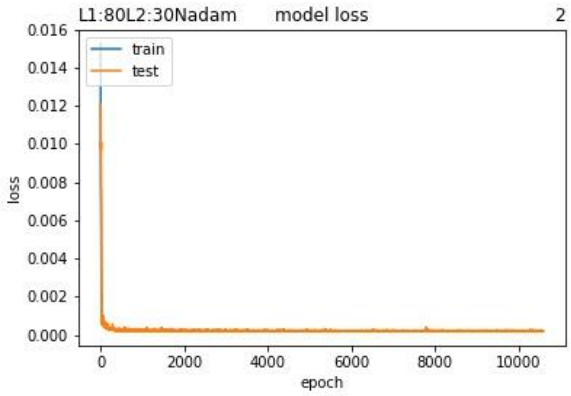
Adagrad	**Tiempo [min]**	**Épocas**	**Adadelta**	**Tiempo [min]**	**Épocas**
Temporada 1	105,40	750	Temporada 1	78,87	1000
Temporada 2	53,53		Temporada 2	74,02	
Temporada 3	82,78		Temporada 3	74,36	

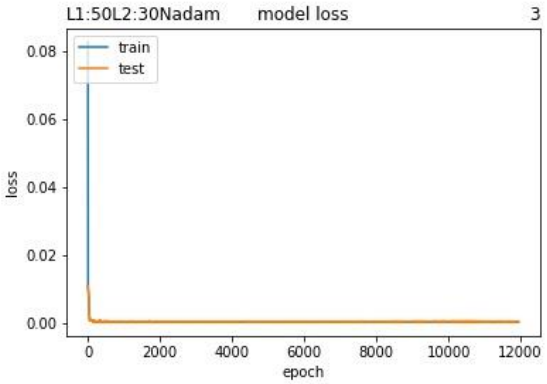
5. Adam	6. Adamax
	
	
	

Adam	**Tiempo [min]**	**Épocas**	**Adamax**	**Tiempo [min]**	**Épocas**
Temporada 1	87,50	750	Temporada 1	70,75	1000
Temporada 2	78,52		Temporada 2	74,04	
Temporada 3	71,04		Temporada 3	58,71	

7. Nadam



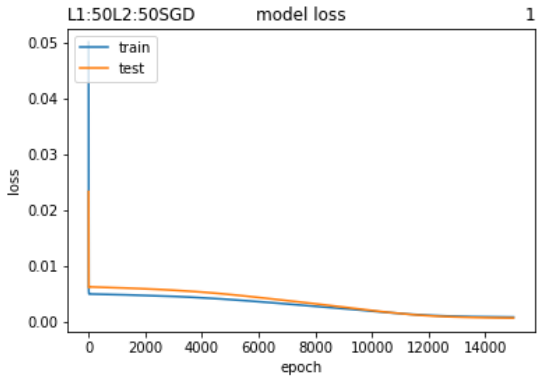


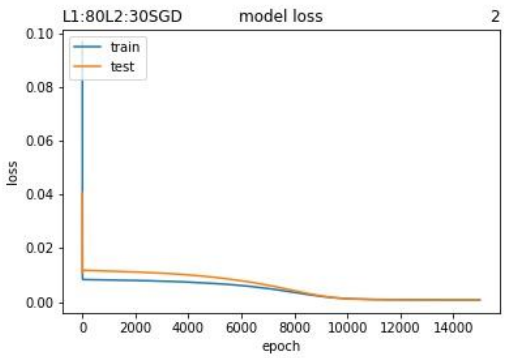


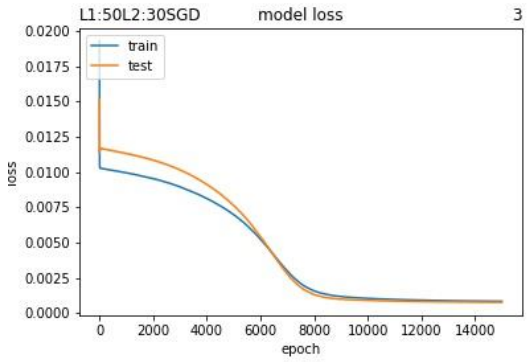
Nadam	Tiempo [min]	Épocas
Temporada 1	67,33	1000
Temporada 2	55,19	
Temporada 3	50,88	

ANEXO F: Estabilización del error de entrenamiento y validación en épocas y tiempo

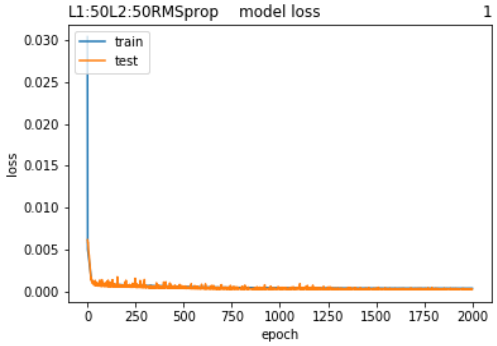
1. SGD

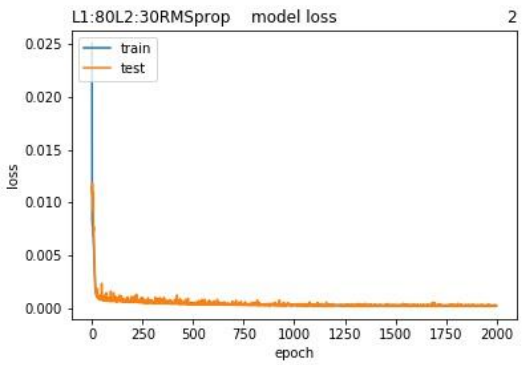


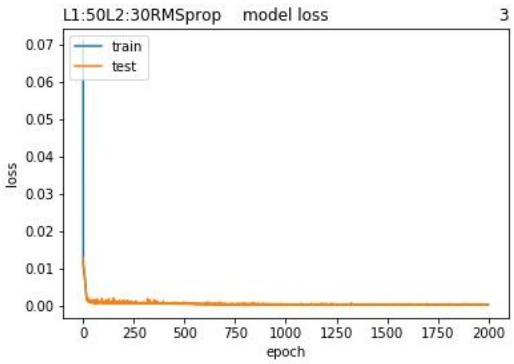




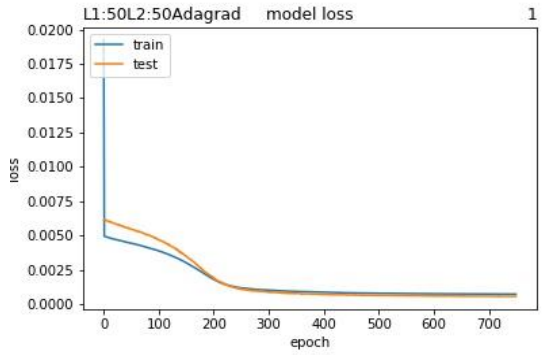
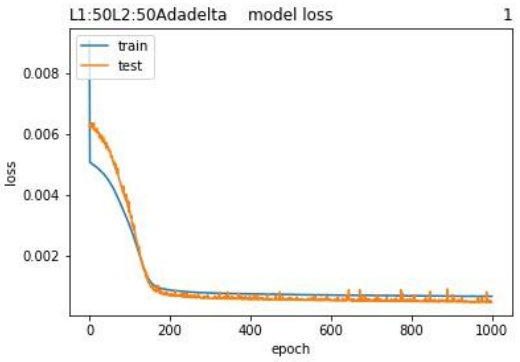
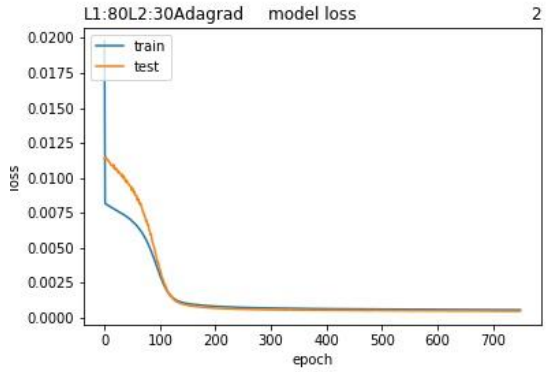
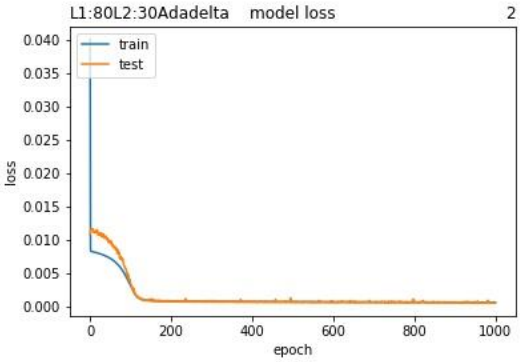
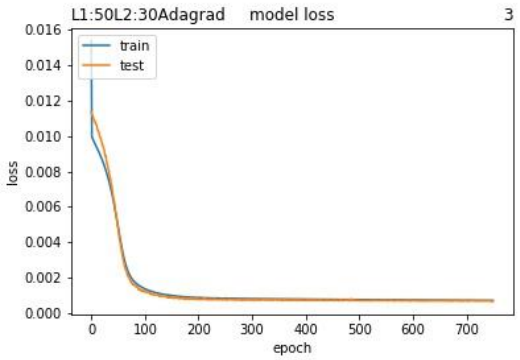
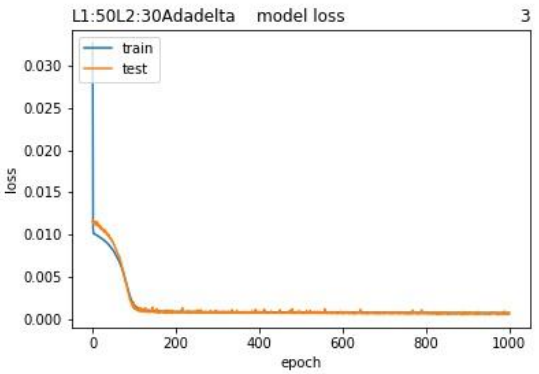
2. RMSprop



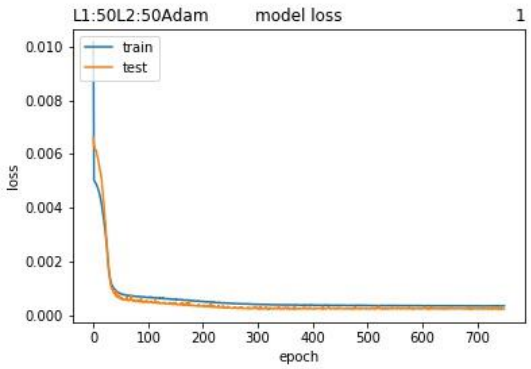
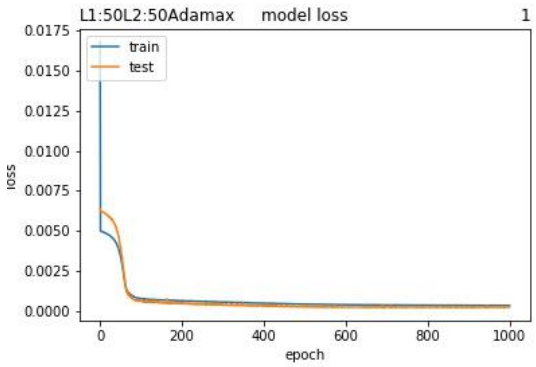
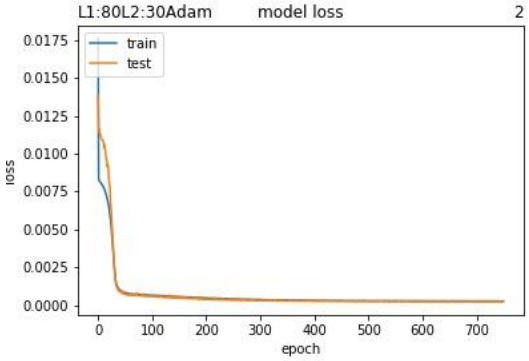
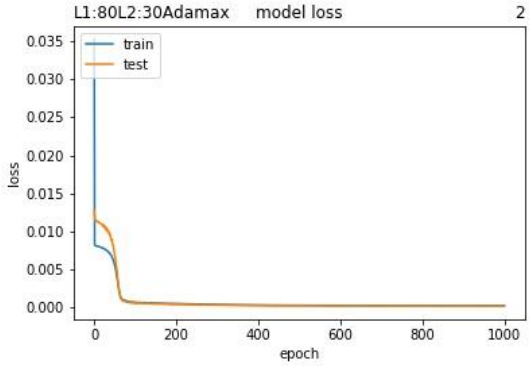
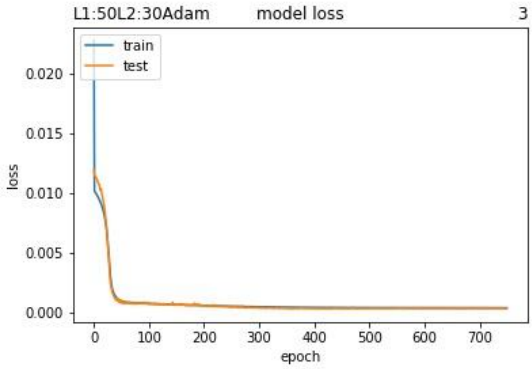
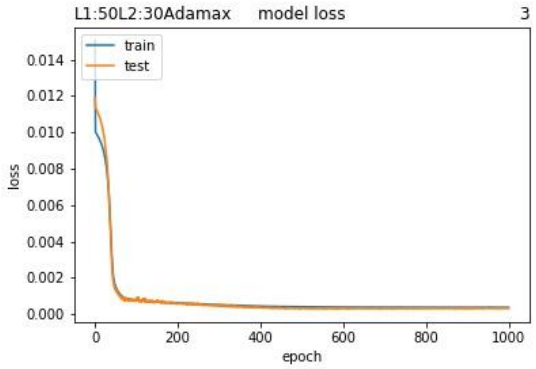




SGD	Épocas	Tiempo [min]	RMSprop	Épocas	Tiempo [min]
Temporada 1	14000	88.8	Temporada 1	1750	11.2
Temporada 2	12000	42.9	Temporada 2	1250	5.4
Temporada 3	10000	43.8	Temporada 3	1000	4.2

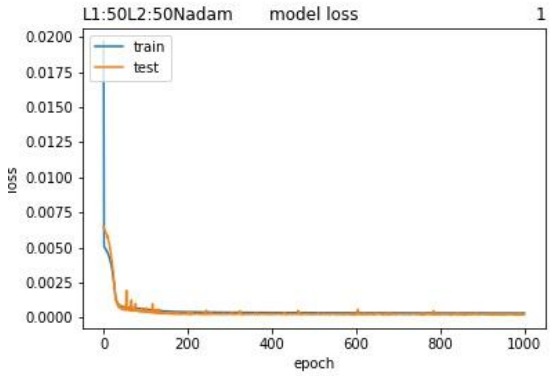
3. Adagrad	4. Adadelta
	
	
	

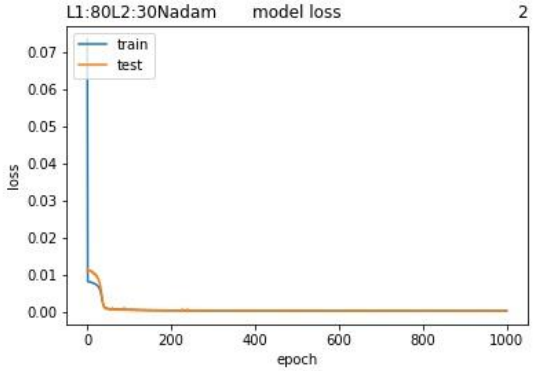
Adagrad	**Épocas**	**Tiempo [min]**	**Adadelta**	**Épocas**	**Tiempo [min]**
Temporada 1	500	3.9	Temporada 1	600	3.9
Temporada 2	250	1.2	Temporada 2	250	1.2
Temporada 3	200	0.8	Temporada 3	200	0.8

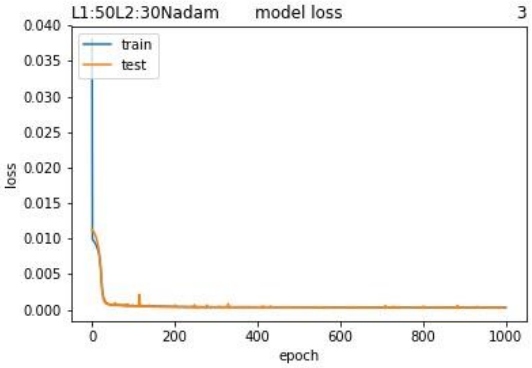
5. Adam	6. Adamax
	
	
	

Adam	**Épocas**	**Tiempo [min]**	**Adamax**	**Épocas**	**Tiempo [min]**
Temporada 1	300	2.6	Temporada 1	400	3.2
Temporada 2	150	0.8	Temporada 2	200	1.3
Temporada 3	300	1.4	Temporada 3	400	2.3

7. Nadam







Nadam	Épocas	Tiempo [min]
Temporada 1	300	2.5
Temporada 2	200	1.2
Temporada 3	200	1.2