Red Neuronal Artificial (RNA) para pronóstico de intensidad de lluvia a 10 minutos en el Área Metropolitana de Guadalajara.

Nancy Rebeca Amador Zaragoz[1] ***Asesor 1:****Alma Delia Ortiz Bañuelos*[1]

[1] *Departamento de Física, CUCEI, Universidad de Guadalajara*
Blvd. Marcelino García Barragán 1421, Col. Olímpica, Guadalajara Jal., C. P. 44430, México
2

Resumen

Se diseñó una Red Neuronal Artificial (RNA) tipo Multilayer FeedForward (MFFN) para pronosticar la intensidad de lluvia a 10 minutos en el Área Metropolitana de Guadalajara. Los datos utilizados durante el entrenamiento fueron los de la estación meteorológica del Instituto de Astronomía y Meteorología (IAM) del año 2019. El propósito de la red es pronosticar lluvia nula, fuerte y torrencial. Se entrenó mediante el algoritmo de retropropagación del error Levenberg-Marquardt mediante la paquetería de aplicaciones de MATLAB. Se llegó a obtener un error cuadrático medio (MSE) de 1.13×10^{-14}, en la etapa de prueba y validación el 100 % de los datos fue clasificado satisfactoriamente. Se realizaron siete pruebas con datos externos al entrenamiento y validación de la red. En esta nueva etapa de prueba la red dio el pronóstico correcto en el 70 % de los casos.

Introducción

Las tormentas son el resultado de la convección de las nubes. El ingrediente principal para este fenómeno es el aire húmedo, más cálido que su entorno, que asciende debido a su menor densidad respecto del aire que lo circunda. Cuando la parcela de aire ascendente se enfría hasta su punto de saturación, se forma un cúmulo, es decir, una célula convectiva.[1]

Si ésta se enfría más, entonces se condensa. Entonces se convierte en lluvia cuando las gotitas se hacen más gruesas y más pesadas.

Estos cambios en las variables de presión, humedad, temperatura etc que se tienen respecto la altura, también se presentan en superficie. Frecuentemente se percibe ese viento arrachado minutos antes de que comience a llover, el descenso de temperatura, aumento de humedad y nubosidad. Por supuesto, estos patrones varían según la tormenta, [2].

Los parámetros físicos son medidos por las estaciones meteorológicas, éstas registran su comportamiento antes, durante y después de la tormenta. Debido a la variación en su comportamiento, en este trabajo se diseña una red neuronal que aprenda de estas fluctuaciones y así obtener un

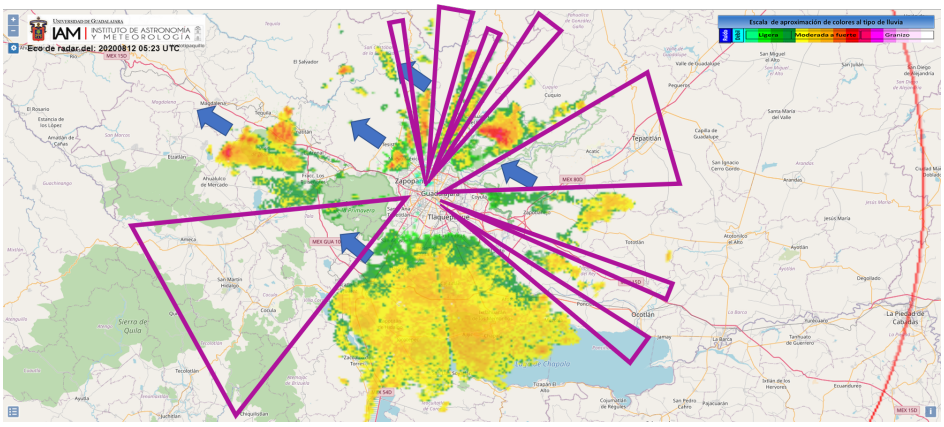


Figura 1: En las áreas encerradas con la línea magenta se encuentran las zonas de obstrucción al radar

pronóstico más preciso a corto plazo mediante los datos 30, 20 y 10 minutos atrás de las estaciones meteorológicas del Área Metropolitana de Guadalajara (AMG), Jalisco.

Debido a la complicada topografía e infraestructura del AMG, sumado a otros factores, en cada temporal lluvioso se tienen afectaciones graves por inundaciones y deslaves (figura 2).

Para realizar el pronóstico a corto plazo de intensidad de lluvia se utilizan los modelos numéricos atmosféricos que se ejecutan cada 4 horas (Global Forecast System "GFS") o 12 horas (European Centre for Medium-Range Weather Forecasts .ECMWFz Met Office weather forecasts for the UK ÜKMO"),[3] estos brindan estimaciones de parámetros meteorológicos con intervalos de 6, 12 o 24 horas y con resolución espacial de 50 a 150 km. Si bien son funcionales para el pronóstico diario, cuentan con ciertas deficiencias, por ejemplo, que no es posible saber si los próximos minutos la tormenta en turno se disipará o continuará con eventos de lluvia intensos y qué valores de precipitación acumulada se espera tener. Una herramienta que brinda apoyo en esta problemática es el radar doppler, que nos permite hacer un barrido vertical (Range Height Indicator RHI") para ver si la tormenta está en etapa de crecimiento, madurez o disipación y así saber si aumentará o disminuirá su intensidad. Una desventaja es que no obtenemos del radar un valor numérico de la cantidad de lluvia en milímetros los próximos minutos, solo si nos permite saber si se espera que se disipe la tormenta o se intensifique. Dicho instrumento es una buena herramienta para el pronóstico a corto plazo, pero no es suficientemente efectiva cuando se trabaja fuera de la oficina pues que no es posible la manipulación del software y en caso de lograrlo, no brinda información sobre varias zonas del AMG en las que debido a los edificios o montañas se obstruye el haz del radar (figura 1).

Este trabajo se propone dar solución a dichas deficiencias y así poder brindar un mejor servicio a los institutos encargados de tomar decisiones para el bienestar público (Protección Civil, Ayuntamiento, CONAGUA y SIAPA), sector privado y a la población en general que requiera de un pronóstico a corto plazo (los próximos 10 minutos).

En la actualidad, se ha implementado el uso de redes neuronales en distintas áreas de las ciencias, una de ellas es la meteorología en el área de pronostico. Lin y Chen en 2004 [4] comparan

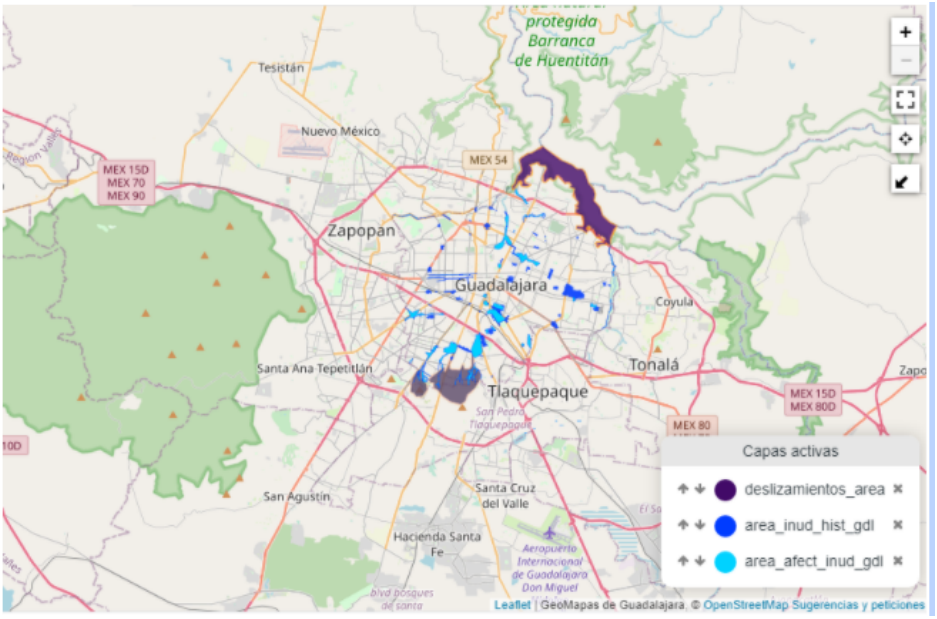


Figura 2: Zonas de frecuente inundación del Área Metropolitana de Guadalajara remarcadas en azul.

diferentes métodos de pronóstico y concluyen que el uso de Redes Neuronales Artificiales (RNA) conduce a mejores resultados.

Metodología

Las redes neuronales artificiales son un modelo inspirado en el funcionamiento del cerebro humano. Está formado por un conjunto de nodos conocidos como neuronas artificiales que están conectadas y transmiten señales entre sí. Estas señales se transmiten desde múltiples entradas hasta generar una o varias salidas.
Las redes neuronales fabrican un sistema complejo de funciones que establecen una relación entre los datos de entrada y salida. Esto, por ejemplo, con la intención de poder predecir los datos de la salida dados ciertos valores de entrada. Durante la creación de una red neuronal para el pronóstico hay tres procesos clave:

1. Entrenamiento
2. Validación
3. Prueba

El set de datos con el que se entrenará es de suma importancia. Si son muy pocos datos el modelo no será lo suficientemente flexible para encontrar una generalización. Si hay un exceso de datos el modelo solo se ajustará a aprender los datos particulares y será incapaz de reconocer datos nuevos,[5].

Algorithms	Update Rules	Convergence	Computation Complexity
EBP algorithm	$w_{k+1} = w_k - \alpha g_k$	Stable, slow	Gradient
Newton algorithm	$\mathbf{w}_{k+1} = \mathbf{w}_k - \boldsymbol{H}_k^{-1}\boldsymbol{g}_k$	Unstable, fast	Gradient and Hessian
Gauss–Newton algorithm	$\mathbf{w}_{k+1} = \mathbf{w}_k - \left(\boldsymbol{J}_k^T\boldsymbol{J}_k\right)^{-1}\boldsymbol{J}_k\mathbf{e}_k$	Unstable, fast	Jacobian
Levenberg–Marquardt algorithm	$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - \left(\boldsymbol{J}_k^T\boldsymbol{J}_k + \mu\boldsymbol{I}\right)^{-1}\boldsymbol{J}_k\boldsymbol{e}_k$	Stable, fast	Jacobian
NBN algorithm [08WC][a]	$\mathbf{w}_{k+1} = \mathbf{w}_k - \boldsymbol{Q}_k^{-1}\boldsymbol{g}_k$	Stable, fast	Quasi Hessian[a]

Figura 3: Especificaciones de diferentes algoritmos

Respecto a la topología de la red, el mejor desempeño para una serie de tiempo se muestra usando una red de tipo MFFN (multilayer feedforward network) [6].Las redes neuronales feedforward son las redes donde las conexiones entre neuronas en capas no forman un ciclo. Lo que significa que la entrada se propaga solo en la dirección de avance (de la capa de entrada a la capa de salida). Este tipo de red utiliza un algoritmo de entrenamiento de retropropagación del error (Backpropagation). El método emplea un ciclo propagación – adaptación de dos fases. Una vez que se ha aplicado un patrón a la entrada de la red como estímulo, este se propaga desde la primera capa a través de las capas siguientes de la red, hasta generar una salida. La señal de salida se compara con la salida deseada y se calcula una señal de error para cada una de las salidas. Las salidas de error se propagan hacia atrás, partiendo de la capa de salida,hacia todas las neuronas de la capa oculta que contribuyen directamente a la salida. [7]

La lenta convergencia del método de descenso más empinado puede mejorarse en gran medida con el método Gauss-Newton algoritmo . Usar derivadas de segundo orden de la función de error para evaluar "naturalmente"la curvatura de superficie de error, el algoritmo de Gauss-Newton puede encontrar tamaños de paso adecuados para cada dirección y converger muy rapido; especialmente, si la función de error tiene una superficie cuadrática, puede converger directamente en el primera iteración. Pero esta mejora solo ocurre cuando la aproximación cuadrática de la función de error es razonable. De lo contrario, el algoritmo de Gauss-Newton sería en su mayoría divergente.
El algoritmo de Levenberg-Marquardt, El algoritmo de Levenberg-Marquardt combina el método de descenso más empinado y el de Gauss-Newton algoritmo. Afortunadamente, hereda la ventaja de velocidad del algoritmo de Gauss-Newton y la estabilidad del método de descenso más empinado. Es más robusto que el algoritmo de Gauss-Newton, porque en muchos casos puede converger bien incluso si la superficie de error es mucho más compleja que la situación cuadrática. Aunque el algoritmo de Levenberg-Marquardt tiende a ser un poco más lento que el algoritmo de Gauss-Newton (en situación convergente), converge mucho más rápido que el método de descenso más empinado. En la figura **??** se ilustran las formas generales del cálculo de pesos para cada algoritmo y la complejidad computacional de calculo que requieren.
La idea básica del algoritmo de Levenberg-Marquardt es que realiza un proceso de entrenamien-

to combinado: alrededor del área con curvatura compleja, el algoritmo de Levenberg-Marquardt

cambia al más empinado algoritmo de descenso, hasta que la curvatura local sea la adecuada para realizar una aproximación cuadrática; entonces aproximadamente se convierte en el algoritmo de Gauss-Newton, que puede acelerar la convergencia de manera significativa. [8]
Al igual que los métodos cuasi-Newton, el algoritmo de Levenberg-Marquardt fue diseñado para

acercarse a la velocidad de entrenamiento de segundo orden sin tener que calcular la matriz de Hesse. Cuando la función de rendimiento tiene la forma de una suma de cuadrados (como es típico en el entrenamiento de redes de retroalimentación), entonces la matriz de Hesse se puede aproximar como:

$$\mathbf{H} = \mathbf{J}^T\mathbf{J} \tag{1}$$

y el gradiente se puede calcular como

$$\mathbf{g} = \mathbf{J}^T\mathbf{e} \tag{2}$$

donde **J** es la matriz jacobiana que contiene las primeras derivadas de los errores de red con respecto a los pesos y bias, y **e** es un vector de errores de red. La matriz jacobiana se puede calcular mediante una técnica de retropropagación estándar que es mucho menos compleja que calcular la matriz hessiana.

El algoritmo de Levenberg-Marquardt utiliza esta aproximación a la matriz de Hesse en la siguiente forma similar a Newton:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \left[\mathbf{J}^T\mathbf{J} + \mu\mathbf{I}\right]^{-1}\mathbf{J}^T\mathbf{e} \tag{3}$$

Cuando el escalar μ es cero, este es solo el método de Newton, utilizando la matriz hessiana aproximada. Cuando μ es grande, se convierte en descenso de gradiente con un tamaño de paso pequeño. El método de Newton es más rápido y más preciso cerca de un mínimo de error, por lo que el objetivo es cambiar hacia el método de Newton lo más rápido posible. Por lo tanto, μ se reduce después de cada paso exitoso (reducción en la función de desempeño) y se incrementa solo cuando un paso tentativo aumentaría la función de desempeño. De esta forma, la función de rendimiento siempre se reduce en cada iteración del algoritmo.
En lo que respecta a la arquitectura de la red neuronal, en diferentes investigaciones se ha estudiado formas de decidir el número de neuronas, aun así, a la fecha, no hay una forma definitiva de resolver este problema. [9]

Un número de neuronas mayor que la necesaria desemboca en un sobreentrenamiento, si son muy pocas el modelo no es suficientemente flexible para soportar cierto grado de no-linealidad [10] Es aconsejable considerar al menos tres valores diferentes para cada parámetro (de ser posible) para permitir la determinación de la curva de respuesta desde el principio. [9]

Para el uso del método de las RNA en series de tiempo es necesario hacer la hipótesis de que la precipitación es un proceso Markoviano [11]. Un proceso de Markov es una serie de experimentos en que cada uno tiene m posibles resultados, E1, E2.....Em, y la probabilidad de cada resultado depende exclusivamente del que se haya obtenido en los experimentos previos. En ese contexto, la altura de lluvia en un momento y lugar dados es vista como una función de un conjunto finito de valores previos. Tal como sugiere la siguiente ecuación

$$x\left(t+1\right) = f\left(x(t), x(t-1), x(t-2)...x(t-k+1))\right) + e(t) \tag{4}$$

Dado que es necesario usar al menos tres datos de entrada distintos, se puede usar los datos de precipitación que provengan de otras estaciones meteorológicas. Trabajos como el de Cisneros en 2007 [12] utilizan como variables de entrada para la predicción del tiempo $t+1$ las variables t, $t-1$,$t-2$ de cinco diferentes estaciones.

En su estudio se emplearon datos de episodios lluviosos dentro de la temporada de lluvias que tuvieran una duración superior a 1 hr y 45 min, concluyendo que intervalos de tiempo de 10 min podrían mejorar el pronóstico y que también podrían ser útiles datos tales como la presión atmosférica, viento o temperatura.

Por lo anterior, en este trabajo se seleccionó el set de datos climatológicos de la estación meteorológica del Instituto de Astronomía y Astrofísica (IAM) del año 2019. El intervalo de tiempo entre cada vector de datos es de 10 minutos.

De los datos del 2019 de la estación del IAM son 52,464 de lo cuales solo 516 son datos de lluvia. Se graficaron los datos de lluvia como se muestra en el conjunto de figuras 4,5 donde mediante círculos se indica el valor del parámetro de dos días en los que llueve. Como se puede observar, no es suficiente el contraste entre valores para hacer un pronóstico visual de la intensidad de lluvia.

Se dividieron los datos en tres sets con al finalidad de entrenar la red:

- No lluvia
- Lluvia fuere
- lluvia torrencial

formando así tres matrices distintas de datos. Para cada una de las matrices existían las siguientes variables en columna.

- Temperatura
- Humedad
- Punto de rocío
- Velocidad del viento
- Presión barométrica
- Temperatura afuera (Temp Out)
- Radiancia solar
- Rain rate

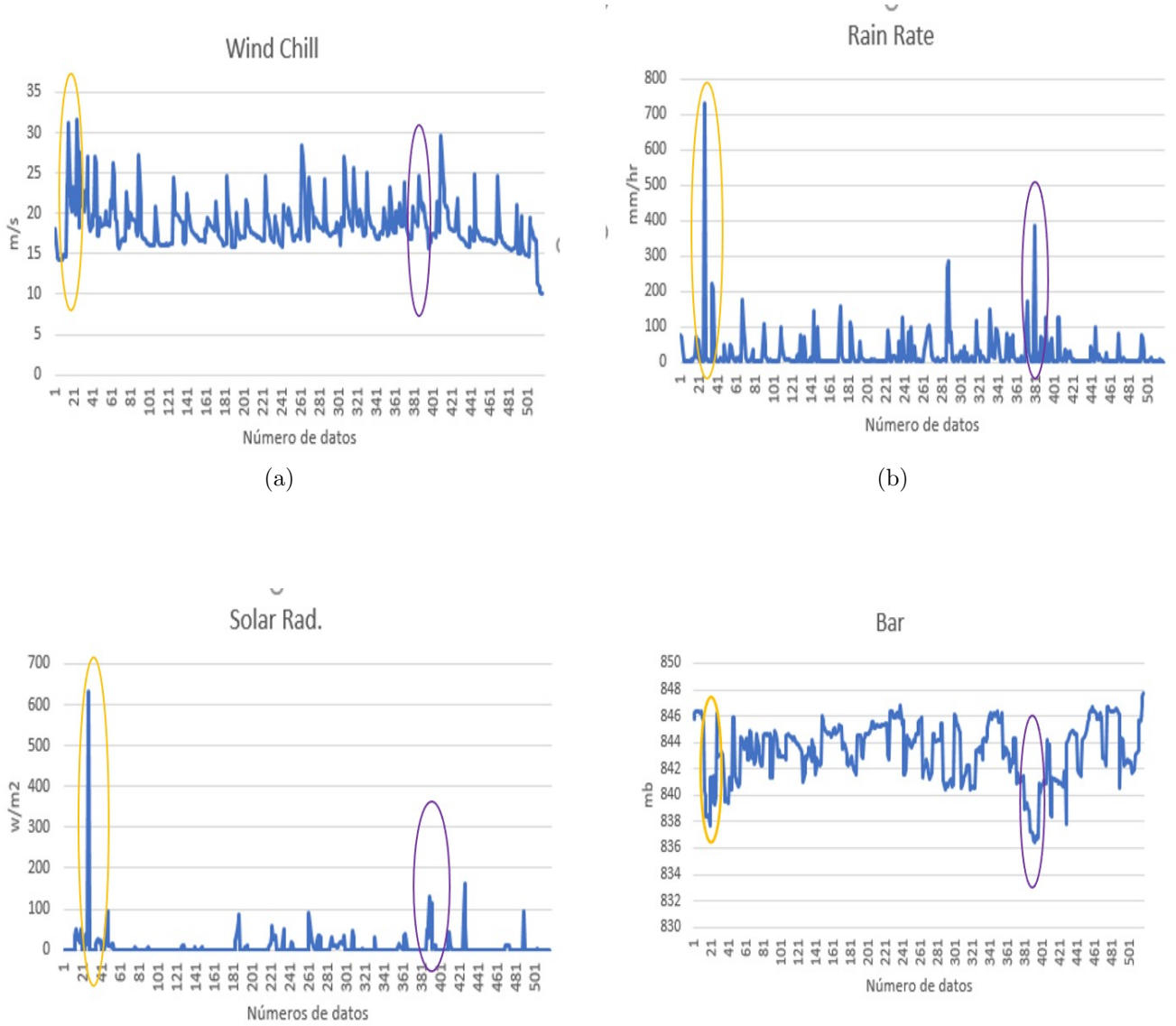
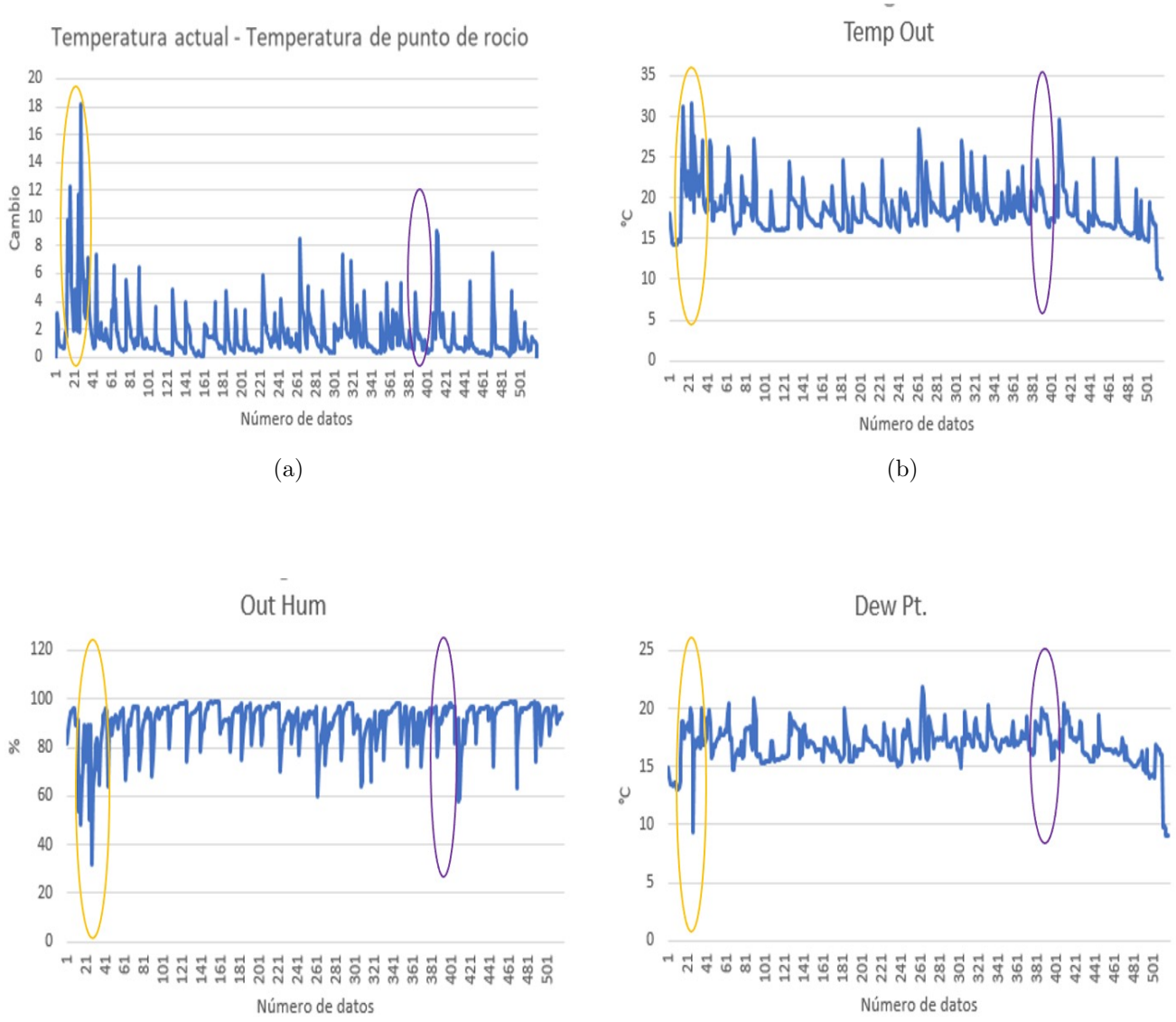


Figura 4: En los circulos amarillos observamos un cambios drásticos en los parámetros físicos, sin embargo, en los círculos morados donde también se presentó una lluvia fuerte según el rain rate (grafica b) no se muestra un comportamiento sobresaliente respecto a los días donde no llovió fuerte. Debido a este ligero cambio de las variables meteorológicas que puede propiciar una lluvia significativa o no significativa, es que se consideró apropiado usar matrices con la mayor cantidad de datos para entrenar la RNA cuando no llueve, cuando llueve fuerte y cuando es torrencial



(a) (b)

(c) (d)

Figura 5: Análogo a la figura 4, se resaltan en círculos color amarillo y morado los valores de las variables meteorológicas durante dos eventos de lluvia distintos. Podemos observar en unos días hay un comportamiento sobresaliente, en otros permanece bastante mediano.

Intensidad	Tamaño de la Matriz	Código
Lluvia nula o ligera	8 x 23	1 0 0
Lluvia fuerte	8 x 28	0 1 0
Lluvia torrencial	8 x 21	0 0 1

Cuadro 1: Tamaño de las matrices de entrenamiento y código numérico que recibió cada vector de datos según su intensidad en la matriz de objetivos (targets)

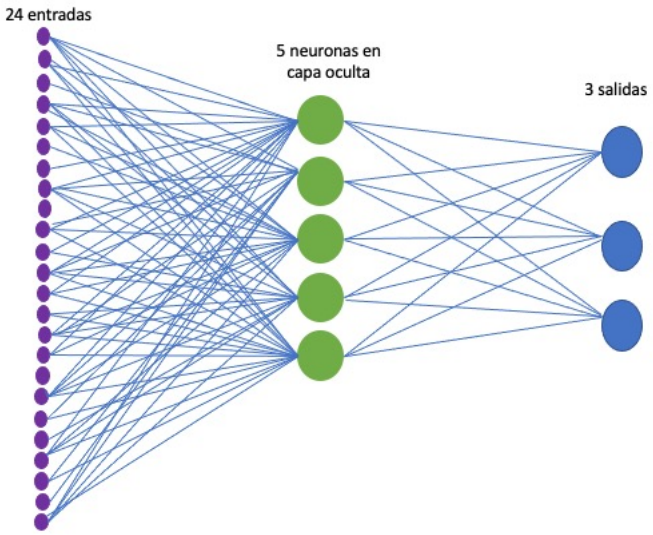


Figura 6: Esquema general de la red neuronal

Se expresan en la tabla 1 los tamaños de las matrices

Se construyó una red tipo Multilayer FeedForward (MFFN) en MATLAB.

Para cada uno de los 8 parámetros se seleccionaron 3 datos diferentes de cada set correspondientes a los tiempos $t, t-1, t-2$. Por lo que resultaría en 24 entradas.
Se quitaron los últimos 10 renglones de cada matriz para dejarlos para la validación.

Al no haber una forma especifica de crear una red neuronal se probó con la siguiente configuración:

- 24 entradas
- 5 neuronas de capa intermedia
- 3 neuronas de salida

En la figura 6 se ilustra esquema de la red. La red tenía como objetivo clasificar las lluvias en nulas, fuertes y torrenciales, por lo que se asignó a cada vector de datos un código numérico según el tipo de lluvia. El código numérico se fabricó para cada set de datos y se guardó como una matriz de objetivos (targets) con los cuales la red se entrenaría. La clasificación queda como en la tabla 1

Como algoritmo de entrenamiento utilizamos el algoritmo de Levenberg-Marquadt. Como funciones de activación para ambas capas se eligió la función Tangente Sigmoidea. La función

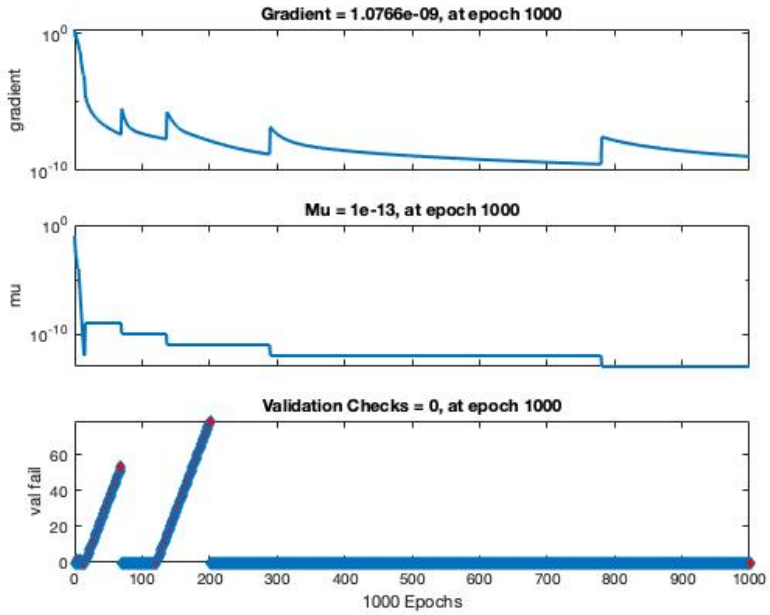


Figura 7: El factor de adaptatividad fue de 1.0×10^{-13} en la epoch no.1000. El minimo local para la función de perdida tuvo un valor de 1.06×10^{-9}.

tangente hiperbólica transforma los valores introducidos a una escala (-1,1), donde los valores altos tienen de manera asintótica a 1 y los valores muy bajos tienden de manera asintótica a -1. Los pesos iniciales así como los bias se asignaron de manera aleatoria. Para el factor de adaptatividad mu (μ) se estableció un factor de decremento de 0.1 y uno de incremento de 10. De forma inicial indicamos que realizara 1000 epochs y que el error máximo permitido fuera de 1×10^{-29}

Resultados

La red realizó 1000 epochs en la etapa de entrenamiento. En el algoritmo de Levenberg-Marquadt es necesario especificar un factor mu (μ) de adaptatividad inicial. Se le dio un valor inicial de 0.1 y concluido el entrenamiento dicho valor fue de 1.0×10^{-13} tal como se puede apreciar en la figura 7. Se logró un error de 1.6×10^{-6} antes de que el comportamiento de la red comenzara a sobreentrenarse como puede apreciarse en la figura 8. El error cuadrático medio (MSE) disminuyo hasta ser de 1.13×10^{-14} durante la etapa de entrenamiento. En las matrices de confusión, 9 podemos notar que el modelo clasifica de forma correcta un 100 % de los datos, tanto en la etapa de entrenamiento como en la de validación.Una matriz de confusión es una herramienta que permite la visualización del desempeño de un algoritmo que se emplea en aprendizaje supervisado. Cada columna de la matriz representa el número de predicciones de cada clase, mientras que cada fila representa a las instancias en la clase real.

Tomamos los datos de siete registros del 2019 que no fueron utilizados para entrenar la red a fin de probar el modelo. En el set de figuras 10 se muestran los resultados. En todos los casos la red debía producir lluvia nula. Predijo con exactitud el pronóstico de 5 de los 7 sets de datos. Es

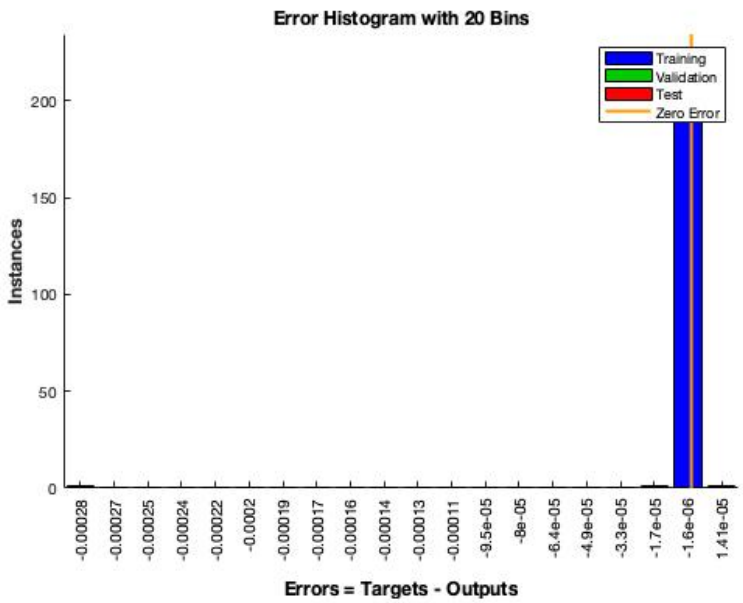


Figura 8: El error entre el dato pronósticado por la red y el dato real para la mayoría de las pruebas fue de 1.6×10^{-6}

decir, tiene un 70 % de exactitud en el pronóstico.

Conclusiones

Resta hacer más pruebas con datos que no se hayan usado ni durante el entrenamiento o la validación a fin de evaluar la exactitud del modelo. Pese a que la red predice la intensidad de lluvia de manera satisfactoria para los datos de 2019 esto no garantiza que el modelo se ajuste a los valores de otros años. Un set de matrices que abarque años anteriores podría mejorar la exactitud de pronóstico de la red. Otro factor de mejora documentado en el articulo de Cisneros en 2007 [12] sería considerar los datos de otras estaciones meteorológicas.

Referencias

[1] UCAR, MetED, "Tormentas locales severas tropicales," https://www.meted.ucar.edu/training_module.php?id=1130#.XicOe2qjnZ4, 2021.

[2] C. D. Ahrens and R. Henson, *Meteorology today: an introduction to weather, climate, and the environment.* Cengage learning, 2021.

[3] Meteolo Beta, "About Weather Models," www.meteolobeta.com, 2021.

[4] G.-F. Lin and L.-H. Chen, "A non-linear rainfall-runoff model using radial basis function network," *Journal of Hydrology*, vol. 289, no. 1-4, pp. 1–8, 2004.

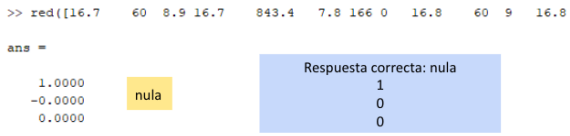
Figura 9: Matrices de confusión del modelo. Durante las etapas de entrenamiento, validación y prueba se muestra que el 100 % de los datos fue clasificado de forma correcta.

(a) Prueba 1

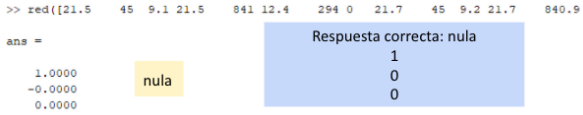
(b) Pruebas 2

(c) Prueba 3

(d) Prueba 4



(e) Prueba 5



(f) Prueba 6

(g) Prueba 7

Figura 10: Vectores de prueba con datos no utilizados ni para entrenar ni para validar la red. En todos los casos el modelo debía predecir lluvia nula

[5] Aprende Machine Learning, "Qué es overfitting y underfitting y cómo solucionarlo," https://www.aprendemachinelearning.com/que-es-overfitting-y-underfitting-y-como-solucionarlo/.

[6] K. C. Luk, J. E. Ball, and A. Sharma, "An application of artificial neural networks for rainfall forecasting," *Mathematical and Computer modelling*, vol. 33, no. 6-7, pp. 683–693, 2001.

[7] R. Rojas, "The backpropagation algorithm," in *Neural networks*. Springer, 1996, pp. 149–182.

[8] C. Lv, Y. Xing, J. Zhang, X. Na, Y. Li, T. Liu, D. Cao, and F.-Y. Wang, "Levenberg–marquardt backpropagation training of multilayer neural networks for state estimation of a safety-critical cyber-physical system," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 8, pp. 3436–3446, 2017.

[9] M. A. Salazar Aguilar, G. J. Moreno Rodríguez, and M. Cabrera-Ríos, "Statistical characterization and optimization of artificial neural networks in time series forecasting: The one-period forecast case," *Computación y Sistemas*, vol. 10, no. 1, pp. 69–81, 2006.

[10] C. Bustos, "Aplicación de redes neuronales al pronóstico de temperatura mínima en la provincia de mendoza."

[11] K. Luk, J. E. Ball, and A. Sharma, "A study of optimal model lag and spatial inputs to artificial neural network for rainfall forecasting," *Journal of Hydrology*, vol. 227, no. 1-4, pp. 56–65, 2000.

[12] H. L. C. Iturbe and I. J. Pelczer, "Redes neuronales artificiales para pronosticar alturas de precipitación cada 15 minutos," *Tecnología y ciencias del agua*, vol. 22, no. 3, pp. 5–21, 2007.