Tarea 2: Redes Neuronales Artificiales

Juan Pablo Arango Atehortúa, Simón Zapata Caro
Antioquia,
Universidad de Antioquia.
juan.arango17@udea.edu.co, simon.zapata@udea.edu.co.

Resumen*—*En este informe, se realiza la resolución de tres problemas de inteligencia computacional, usando redes neuronales artificiales (ANN). El primer ejercicio es la implementación de una ANN entrenada bajo los algoritmos de entrenamiento de mínimos cuadrados (LMS) y regresión lineal para definir la eficiencia óptima de energía en una planta dadas ciertas características. El siguiente problema es un ejercicio de clasificación de 2 clases, utilizando los algoritmos de entrenamiento del pérceptron y de regresión logística orientado a reconocimiento de caracteres. Por último, se implementa una red multicapa (MLP) para dar solución a un problema de clasificación con 12. Para cada uno de los problemas se escogen distintas topologías y parámetros de red y se valida cada arquitectura para evitar problemas de sobre-entrenamiento y escoger la arquitectura de red que optimiza el error

Index Terms—*Artificial Neural Networks, Multi-Layer perceptron, LMS, Regression, Classification, Machine Learning w/ python.*

I. INTRODUCCIÓN

Las redes neuronales artificiales (o ANN, por sus siglas en inglés) son una herramienta de Machine Learning de aprendizaje supervisado, las cuales se han constituido en una de las principales herramientas aplicables desde la inteligencia artificial a problemas de clasificación, regresión, detección de patrones, visión por computadora, entre otras. Su uso es enormemente apetecido debido a su relativa simpleza matemática y su diversa aplicabilidad, y tomó especial auge con el desarrollo del algoritmo de backpropagation (Werbos 1975), que cerró el eslabón faltante para el entrenamiento adecuado de la red para problemas no separables linealmente. Hoy son tal vez uno de los métodos más usados, y conforman la base del estado del arte de la inteligencia computacional moderna: el Deep learning.

Para entender los conceptos más generales de las ANN y mirar su capacidad de modelar cualquier tipo de función, se implementarán tres de estas redes, entrenadas bajo distintos algoritmos (viendo las fortalezas y debilidades de cada uno) utilizando el lenguaje de programación *python* por ser de código abierto y por su rapidez ante problemas de alto costo computacional.
Para poder implementar una ANN, primero se debe contar con una base de datos previamente etiquetada, tanto para que la red aprenda de los datos, como para validar las topologías de cada red y escoger la óptima, en términos de la que reduce al máximo una función de costo/error.

A. ***Problema de regresión utilizando una neurona, entrenada por los métodos de 'regresión lineal' y LMS***

El primer problema a resolver es un problema de regresión, lo que se pretende es estudiar el efecto de ocho (8) características (área superficial, área de las paredes, área del techo, altura promedio, orientación, etc.) de 768 edificaciones residenciales en la cantidad de calor a aplicar a dichas residencias con el fin de aumentar la eficiencia de energía, reducir costos energéticos y construir edificios *sostenibles*.
Para lograr esto, se implementa una ANN de una neurona (ver **Fig. 1**), y se plantea un problema de optimización, en el cual se desea encontrar los pesos de cada entrada (similares a los estímulos de una neurona humana) para los cuales se minimiza una función de error entre los datos que se esperan (Etiquetas o *Labels*) y los que se están prediciendo. El enunciado anterior es aplicable a cada uno de los problemas tratados en este informe.

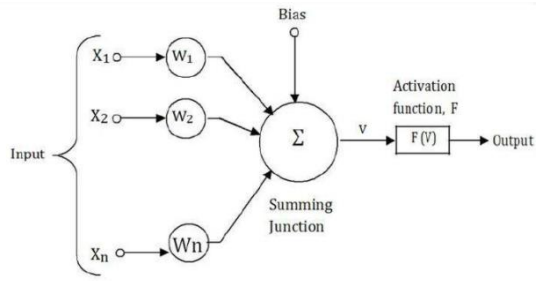
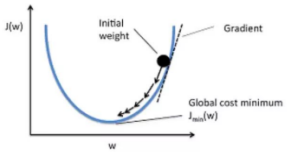


Fig. 1, Arquitectura de ANN de una sola neurona

Para encontrar estos de regresión, se plantean dos heurísticas (métodos iterativos) de solución: utilizando una regresión lineal y utilizando el algoritmo de mínimos cuadrados (LMS), cada uno de estos emplean el gradiente descendiente [4] como algoritmo para minimizar la función de error, el cual no es más que ir ajustando pesos con base en la derivada de la función de error hasta llegar a un óptimo local.



$$w^{(i+1)} = w^{(i)} - \alpha * \frac{dJ(w)}{dw}$$

Fig. 2, Explicación del método del gradiente descendiente

En general, una regresión lineal es un modelo matemático usado para aproximar la relación de dependencia entre una variable dependiente Yi con respecto a unas características de entrada Xi aleatoriamente distribuidas en el espacio.
El algoritmo utilizado para

Repita:

$$h_x = \theta^{\mathrm{T}} X$$

$$E = \frac{1}{2M}(h_x - y) * (h_x - y)^{\mathrm{T}}$$

$$\frac{dE}{d\omega_k} = \frac{1}{M}(h_x - y) * X^{\mathrm{T}}$$

$$\omega_{k+1} = \omega_k - \alpha e \frac{dE}{d\omega_k}$$

Hasta: número de iteraciones o hasta que $E(función\ de\ error)$ no varíe más.

Fig. 3, Algoritmo R.Lineal

Ahora, una modificación de este algoritmo es el LMS (Least mean square), el cual tiene la ventaja de que no necesita un alto costo computacional (uso de GPU y demás sistemas embebidos), y la modificación del algoritmo es la siguiente.

Repita:

Escoja (x_i, y_i)

$$g = \omega_k^T x_i$$

$$e = g - y_i$$

$$\omega_{k+1} = \omega_k - \alpha e x_i$$

Hasta: número de iteraciones o hasta que e no varíe más.

Fig. 4, Algoritmo LMS

La diferencia entre estos dos métodos radica en que la regresión lineal es mucho más rápido que el LMS, ya que en la función de error de esta última no se tienen en cuenta todas las muestras, sino solo una muestra, pero trae una desventaja y es que con la regresión lineal puede no llegar a una solución óptima, aunque esto también depende del número de características, el número de muestras, la calidad de la base de datos de entrenamiento y la semilla del problema, es decir, el punto inicial de los pesos de la Red neuronal.

Entrenamiento de la ANN y validación de cada topología

Para este problema, se plantea como objetivo analizar la diferencia entre ambas rutinas de entrenamiento, cuál es la óptima para el ejercicio, utilizando como métrica el MAE (error absoluto medio)[5], además de cuáles son los efectos en el resultado final si variamos la tasa de entrenamiento() y qué valores de este parámetro resultan en modelos aceptables.

Utilizando una validación bootstrapping entrenando la neurona de procesamiento con *LMS*, utilizando diez (10) iteraciones se corroboran modelos con 30 valores de αentre 10^(-5) y 0.5 para escoger el rango óptimo de este parámetro.

Además, se utiliza la normalización *min-max* para estandarizar los valores de las características entre 0 y 1, para que el efecto de la entrada de bias sea de efecto.

	MAE de entrenamiento	MAE de validación	alpha
0	2.709460	2.721113	0.000010
1	2.487561	2.480490	0.000015
2	2.338527	2.330901	0.000021
3	2.263308	2.257902	0.000031
4	2.238744	2.239046	0.000044
5	2.223260	2.229119	0.000065
6	2.215586	2.226953	0.000094
7	2.209961	2.228882	0.000136
8	2.207465	2.233322	0.000198
9	2.203289	2.238483	0.000287
10	2.196597	2.241580	0.000417
11	2.186218	2.239740	0.000606
12	2.176278	2.234999	0.000880
13	2.170500	2.232310	0.001278
14	2.167746	2.232882	0.001856
15	2.168148	2.236148	0.002695
16	2.170803	2.241678	0.003913
17	2.177560	2.251482	0.005683
18	2.185417	2.262656	0.008253
19	2.195122	2.275499	0.011985
20	2.211049	2.292578	0.017405
21	2.228489	2.312307	0.025276
22	2.252236	2.344376	0.036707
23	2.291938	2.393534	0.053306
24	2.351918	2.454987	0.077412
25	2.460986	2.575232	0.112420
26	2.744116	2.860283	0.163260
27	3.406765	3.513305	0.237090
28	4.645359	4.773944	0.344307
29	NaN	NaN	0.500012

Tabla 1, Resultados de bootstrapping para escogencia de tasa de entrenamiento

	1. alpha	2. error de entrenamiento:	3. error de validación
0	0.000100	14.426164	14.486253
1	0.000215	10.258553	10.381326
2	0.000464	7.141220	7.154593
3	0.001000	5.383726	5.438845
4	0.002154	3.834720	3.880148
5	0.004642	2.807691	2.913885
6	0.010000	2.365361	2.473039
7	0.021544	2.216635	2.365051
8	0.046416	2.201524	2.380102
9	0.100000	2.175060	2.374729

Tabla 2, Resultados de cross-validación para encontrar el modelo óptimo en Regresión Lineal

Rutina de entrenamiento	**MAE de testeo**
Regresión Lineal	2.469444845086216
LMS	2.085343783026517

Tabla 3, Resultados de cross-validación para encontrar el modelo óptimo en LMS

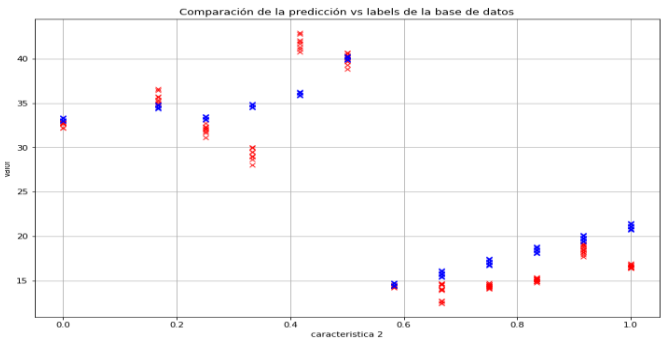


Fig. 5, Representación 2D aprox. de la información

Análisis de resultados

Para este problema, se puede apreciar que, utilizando tasas de entrenamiento de entre 0.046416 y 0.1000, se obtiene modelos que oscilan entre 2.1 y 2.9 unidades de error absoluto, en lo que podemos concluir que se pueden escoger estas tasas de entrenamiento el resultado de predicción tendrá un valor aceptable, ahora, escogiendo el parámetro con menor MAE de validación, se calcula el error de testeo final de cada modelo, entrenado por regresión lineal y por LMS, y **se obtiene que el error en LMS, con respecto a la regresión lineal es menor**, lo que se esperaba, debido a que la función de error del algoritmo LMS depende de cada muestra, lo que calibra de una mejor manera cada peso, en orden de obtener una menor función de error.

Ahora, también se realiza una representación cartesiana de los datos etiquetados v.s. los datos predichos para el modelo ANN (entrenado bajo LMS, **Fig. 5**) y se logra distinguir la estimación de nuestro sistema inteligente, donde la predicción son los puntos de color azul; se puede decir que la precisión de la predicción es alta y los datos del problema de regresión no se dispersan demasiado con respecto a los datos de referencia esperados.

B. Problema de clasificación 2-clases utilizando una neurona, entrenada por los métodos de 'regresión logística' y 'Perceptrón'

Entrenamiento de la ANN y validación de cada topología

La base de datos está compuesta por 16 de atributos de una imagen en blanco y negro cada una asociada a una letra del abecedario, el objetivo es clasificarlas en 2 grupos: aquellas pertenecientes al conjunto {A,R,N,G,O,Z,P,T} y aquellas que no pertenezcan a él.
Se implementó una red neuronal compuesta por una neurona entrenada mediante el algoritmo de *'pérceptron'*, éste es ampliamente utilizado para problemas de clasificación de 2 clases, se caracteriza por codificar los labels como -1 y 1 correspondiente a cada clase, tener una función de error de la forma: $E(w) = -\sum_{x_i \epsilon M} (w^T x_i) y_i$, $M = \{x_i : (w^T x_i) y_i < 0$ y porque actualiza los pesos con base en las muestras que fueron mal clasificadas únicamente (**Fig. 6**), esto puede hacer que la función de error oscile abruptamente debido a que al actualizar los pesos de la red de forma que mejore la clasificación de una muestra que había sido mal clasificada puede hacer que deje de clasificar correctamente otras muestras que habían sido clasificadas de manera adecuada previamente, además, la forma de codificar las etiquetas de los labels hace que la validación de la predicción se pueda realizar mediante la multiplicación de esta con su label, esto es $x_i y_i > 0$para muestras bien clasificadas. además se utilizó la función sigmoidal como función de activación de la neurona, esta ha demostrado dar buenos resultados en problemas de clasificación en general, debido a que es, en su forma, similar al escalón pero entrega valores análogos (infinitos) en el intervalo [0,1](Se puede cambiar el rango mediante escalamiento y traslación). Para implementar el perception se hizo necesario codificar las etiquetas de la clase 1 con el valor de 1 y la clase 2 con

el valor de -1. Se normalizaron los datos mediante min-max,, se tomó el 15% de los datos para testing, El 85% restante se utiliza en el método de validación cruzada para entrenamiento y validación, utilizando 10 folds, se calculó la tasa de entrenamiento y validación para esta. Se seleccionó la configuración de red (vector de pesos y parámetros tales como la función de activación óptima) a la cual pertenecía el mínimo valor de la función de error para realizar el testeo, se calculó su efectividad.

repeat

Escoja $(\mathbf{x}_i, y_i)$

$g = \mathbf{w}_k^T \mathbf{x}_i$

$e = g - y_i$

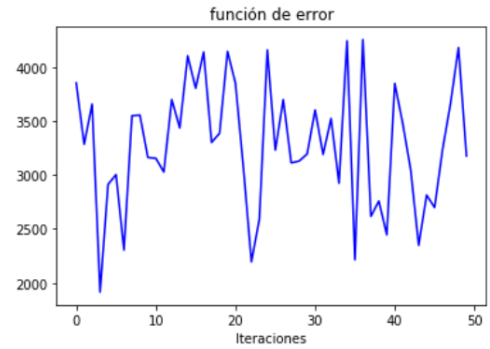
$\mathbf{w}_{k+1} = \mathbf{w}_k - \mu e \mathbf{x}_i$

until Condición de terminación.

Fig. 6, Algoritmo Perceptrón.

Utilizando el mismo conjunto de datos, se entrena otro modelo de clasificación (ANN de una neurona), esta vez utilizando el algoritmo de **Regresión Logística**, que es un análisis estadístico inferencial de la información, el cual trata, grosso modo, de minimizar una función de error, definida en términos de las probabilidades de que cada muestra pertenezca a sendas categorías.

El algoritmo es similar al de regresión lineal, la diferencia consiste en escoger el modelo óptimo con base en este nuevo error, que se define, además, por la cantidad de información de los datos de cada clase [7]. Se normalizaron los datos mediante Z-Score, se utilizó la función sigmoidal como función de activación de la neurona y bootstrapping como algoritmo de entrenamiento y validación, también se estableció un umbral para decidir cuales pertenecían a la clase 1 y .Al igual que en el caso de perceptrón, se calculó el error de entrenamiento, validación y testeo.




```
iteracción óptima: 3, función de error: 1914.0218906291802
```

Fig. 7, Iteración de validación cruzada, algoritmo perceptrón.

Tasa de entrenamiento	Tasa de validación
71,1601307189542%	71,1058823529411%

Tabla 4, Resultados validación cruzada, algoritmo perceptrón.

Tasa de testeo
63,9333333%

Tabla 5, Efectividad red óptima encontrada, algoritmo perceptrón.

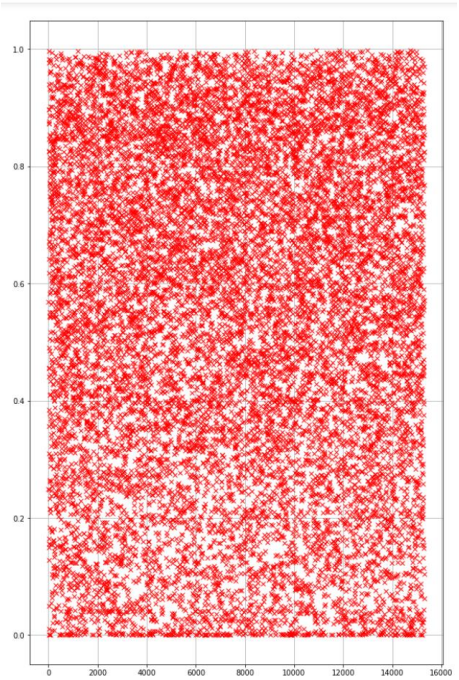


Fig. 8, predicción de validación, algoritmo regresión logística.

Validación	Función de error
1	1.1130965525174676
2	0.8096584840757964
3	1.0455618400880367
4	0.9298415006973706
5	0.9627544737368643
6	1.1798787855869985
7	1.1203002083537568
8	0.8986574653930444
9	0.9733998099609553
10	1.0869885962476087

Tabla 6, función de error en validación cruzada, algoritmo regresión logística.

	1. error de entrenamiento:	2. error de validación
0	0.180654	0.157059
1	0.199085	0.185294
2	0.149346	0.169412
3	0.177582	0.188235
4	0.208627	0.217059
5	0.189085	0.178824
6	0.191242	0.203529
7	0.172614	0.172941
8	0.153660	0.155294
9	0.163529	0.162941

Tabla 7, Resultados validación cruzada, algoritmo regresión logística.

Tasa de testeo
84,1%

Tabla 8, Efectividad red óptima encontrada, algoritmo regresión logística.

Análisis de resultados

En cada una de las iteraciones de la validación cruzada se observa como la función de error oscila abruptamente debido a que los pesos se actualizan con base a las muestras mal clasificadas, tal como se expuso previamente.

La tasa de entrenamiento y validación son muy aproximadas entre sí, lo cual es un indicador de que la red no está sobre entrenada, debido a que al probar la red con datos diferente a los datos de validación obtenemos una efectividad en la red muy aproximada.

La efectividad de la red óptima obtenida en el testeo fue relativamente baja, esto puede deberse a que el número de letras asignado para la clase 1 es 3.375 veces menor que el número de letras de la clase 2, por lo que la probabilidad de que una letra (entrada) pertenezca a la clase 1 es menor que la probabilidad de que pertenezca a la clase 2, esto se refleja en el entrenamiento de la red puesto que al generalizar implícitamente afectará esta probabilidad.

En la **fig.8** se observa la distribución de los datos de salida del algoritmo, el umbral de decisión (0,5 en este caso) permite discernir si la entrada pertenece a la clase 1 o 2, por lo que la efectividad de la red dependerá de la escogencia adecuada del mismo.
El algoritmo perceptrón toma mayor tiempo de ejecución para entrenar la red debido a que deben actualizar los pesos de la red con base a cada una de las muestras que hayan sido clasificadas erróneamente en la iteración anterior.

C. Problema de clasificación multi-clase utilizando un Pérceptron multicapa, entrenado utilizando backpropagation

Cuando abordamos un problema de clasificación de 3 o más clases, no podemos utilizar solo una neurona entrenada por perceptrón, pero podemos unir varias neuronas y aplicando operaciones entre cada salida, para producir el resultado que queremos; en este orden de ideas, se emplea una red neuronal multicapa (MLP, por sus siglas en inglés), como la que se puede visualizar en la **Fig. 9,** donde se tiene una capa de entrada, que lista las características más una neurona de bias; también se tienen unas capas ocultas que procesan la información proveniente de estímulos de neuronas anteriores, es decir, se propaga la información hacia adelante (*Forward*) y al final se organiza el resultado en una capa de salida, que arroja un valor, que refleja el un grado de pertenencia del dato □□a cada una de las clases; al final,

se asigna el dato a la clase cuyo grado de pertenencia sea mayor.

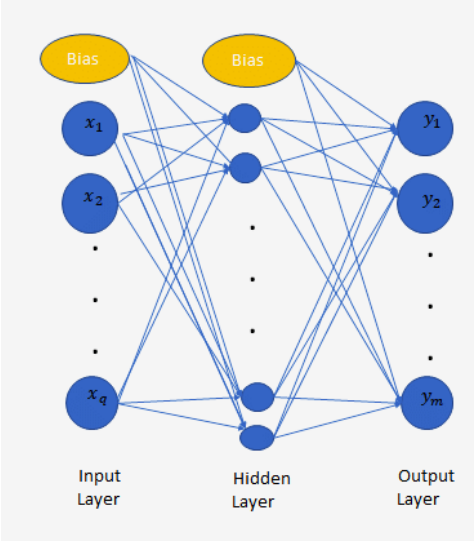


Fig. 9, Diagrama de una (MLP)

Ahora, un problema que se da como consecuencia es el proceso de escoger una topología de red, es decir, cuantas capas ocultas se deben tener y cuantas neuronas por capa, en orden de obtener un error mínimo entre el dato que clasifica la red con respecto a su etiqueta.

El problema que se desea resolver es el siguiente: de nuestra base de datos, , que se compone de 2105 datos, previamente etiquetados, cada uno con 76 características de voz de ciertos animales, se desea obtener un clasificador que permita etiquetar a un animal dentro de 12 clases de animales.
A esta base de datos se le aplicaran las operaciones de normalización (Z-score) y de selección de datos en datos de validación y datos finales de prueba.

Para esto, se utilizará una red neuronal multicapa, y, debido a que para cada problema en específico se debe encontrar la topología que reduzca el error entre la predicción vs. los datos esperados, se escogen ciertas topologías, disponibles en la **Tabla 9,** (aumentando el número de neuronas por capa y el número de capas ocultas), se encuentran los pesos (en este caso, matrices de pesos) que minimicen la función de error y se valida cada topología utilizando una métrica (Tasa de clasificación o Accuracy) y validación bootstrapping para garantizar la generalización de nuestra red, es decir, que tan bien predice esta ante datos nuevos, es decir, diferentes de los datos de entrenamiento (*Notación: [4, 5, 6] sería tener 3 capas ocultas, la primera con 4 neuronas, la segunda con 5 y la tercera con 6, sin contar la neurona de bias, que se cuenta dentro de la rutina de entrenamiento*)

[7]	[7,15]
[15]	[7,31]
[31]	[7,63]
[63]	[15,15]
[127]	[15,31]
[7,7]	[7,7,7]

Tabla 9, Topologías escogidas para validación, MLP

Y para entrenar cada modelo, se utiliza el algoritmo *BackPropagation*, que no es más que definir una función de error, en términos de unas diferencias (o deltas), que se van propagando hacia atrás, hasta llegar a la primera capa, y que a su vez dependen de cada uno de los pesos y de las entradas y salidas de cada neurona de la red.

el algoritmo se puede visualizar en la referencia [1]; y se puede ver el conjunto de *notebooks de python* anexos a este informe, donde se encuentra el código de la rutina de entrenamiento de la MLP, genérico, al cual se le puede variar la topología y cada parámetro de entrenamiento.

Entrenamiento de la ANN y validación de cada topología

```
para sizes =  [7] , da un error de  45.08699954883946
para sizes =  [15] , da un error de  16.647922392048237
para sizes =  [31] , da un error de  11.847478844294072
para sizes =  [63] , da un error de  17.052725535982677
para sizes =  [127] , da un error de  28.172624206608777
para sizes =  [7, 7] , da un error de  241.20039196558045
para sizes =  [7, 15] , da un error de  124.99270639421977
para sizes =  [7, 31] , da un error de  6057.150122587753
para sizes =  [7, 63] , da un error de  8101.499980333355
para sizes =  [15, 15] , da un error de  170.4830646952207
para sizes =  [15, 31] , da un error de  4777.214901594154
para sizes =  [7, 7, 7] , da un error de  321.0155810540021
```

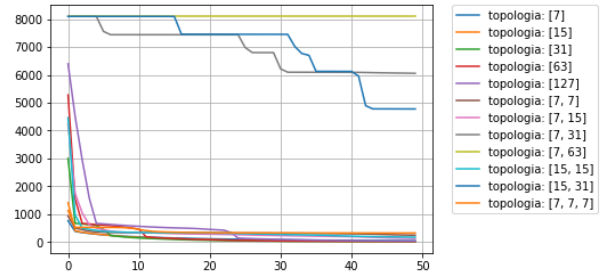


Fig. 10, Resultado de entrenamiento de cada topología

	1.topologia	2. tasa de entrenamiento	3. tasa de validación
0	[7]	0.977597	0.930159
1	[15]	0.995248	0.946032
2	[31]	0.998642	0.949206
3	[63]	0.989138	0.958730
4	[127]	0.981670	0.933333
5	[7, 7]	0.731840	0.742857
6	[7, 15]	0.904956	0.869841
7	[7, 31]	0.520027	0.501587
8	[7, 63]	0.054990	0.069841
9	[15, 15]	0.795655	0.825397
10	[15, 31]	0.520027	0.507937
11	[7, 7, 7]	0.575017	0.565079

Tabla 10, Resultados del bootstrapping en cada red multicapa para encontrar el modelo óptimo

Rutina de entrenamiento	**Error de testeo**
Topología de 1 capa oculta, con 32 neuronas	3.15457 %

Tabla 11, Resultados de cross-validación para encontrar el modelo óptimo en LMS

Análisis de resultados

Si nos remitimos a la **Tabla 10,** se puede apreciar que los menores errores se obtienen utilizando solo una capa, con un número de neuronas que puede oscilar entre 15 y 30 neuronas, y se concluye que no se debe aumentar el número de capas ocultas debido a que aumenta el error de clasificación; también se puede observar que cada modelo no se sobreentrena, independientemente de la topología.

Se escoge la topología de **31 neuronas** (sin contar la neurona de bias), debido a que es la que tiene el error óptimo; su error de clasificación final es del 3,155%.
Ahora, si vemos este error, es bastante pequeño, esto puede deberse a que la red está generalizando muy bien o que la distribución de las muestras tomadas no es lo suficientemente amplia para que la red generalice; puede suceder que para datos distintos de los de la base de datos, atípicos a nuestras muestras, la red no sea capaz de clasificarlos adecuadamente.

II. CONCLUSIONES

Se debe ajustar cada red neuronal artificial a cada problema, escogiendo propiedades de la misma, tanto la topología de red como los parámetros de entrenamiento. Debido a esto, se encuentra para el primer problema una topología que minimiza el error a un valor de 2.0853 MAE, entrenando bajo el algoritmo del LMS, el cual, puede ser óptimo, pero puede acarrear un tiempo de entrenamiento mayor, pero un costo computacional menor, con respecto a la regresión lineal.

Los algoritmos pérceptron y regresión logística permiten realizar una clasificación bi-clase a partir una base de datos, específicamente para la clasificación de las letras planteadas anteriormente el algoritmo de regresión logística obtuvo una tasa de testeo considerablemente mejor que el algoritmo pérceptron, es decir que se espera que esta tenga mejor desempeño a la hora de cumplir su objetivo, además posee un tiempo de entrenamiento menor, lo cual puede ser importante si el número de datos aumentara; sin embargo, no es suficiente esto para concluir que el algoritmo de regresión logística tiene mejor desempeño en general para la clasificación (de 2 clases) respecto a pérceptron, puesto que la eficacia de cada algoritmo depende también de la naturaleza del problema y del número y la calidad de los datos por lo que para otro problema, con una base de datos diferente o incluso una función de activación diferente, el algoritmo pérceptron podría tener una efectividad mayor que la de regresión logística.

Con respecto al tercer punto, se ve la importancia del entendimiento del problema y con base en esto, la escogencia de la topología óptima; para este caso, se obtuvo una ANN con un error extremadamente pequeño, y esto se debió a la calidad de las muestras en la base de datos, lo cual es un factor decisivo a la hora de diseñar un sistema inteligente de éste estilo. además, es importante resaltar que no necesariamente se deben tener muchas neuronas con muchas capas ocultas para obtener siempre buenos resultados o para generalizar correctamente, y allí radica la importancia de validar cada modelo, para evitar escoger una red sobre-entrenada.

III. REFERENCIAS

[1] Isaza, Claudia; Gómez, William E.; *Notas de clase de Fundamentos de Inteligencia Computacional*, módulo de Redes neuronales artificiales, Universidad de Antioquia.

[2] Tsanas, A., & Xifara, A. (2012). Accurate quantitative estimation of energy performance of residential buildings using statistical machine learning tools. *Energy and Buildings*, *49*, 560-567.

[3] Merz, C.J., Murphy., Murphy, P.M.: UCI Repository of Machine Learning Database. Department of Information and Computer Science, University of California, Irvine, CA (1998).
[4] García, J. I. (2011). Algoritmos iterativos: método del gradiente descendente para calcular el mínimo de una función.
[5] Willmott, C. J., & Matsuura, K. (2005). Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance. Climate research, 30(1), 79-82.
[6] Isaza, Claudia; Gómez, William E.; *Notas de clase de Fundamentos de Inteligencia Computacional*, módulo de Selección de modelos, Universidad de Antioquia.
[7]Entropia (Teoria de la información), disponible en:
https://es.wikipedia.org/wiki/Entrop%C3%ADa_(informaci%C3%B3n)